Chair of Applied Mechanics
TUM School of Engineering and Design
Technical University of Munich

Hyper-reduction of Nonlinear Rubber Behaviour using Abaqus and Python with a Co-simulation Approach

Hyper-Reduktion des nichtlinearen Gummiverhaltens mit Abaqus und Python in einem Co-Simulation-Ansatz

Scientific work for obtaining the academic degree
Master of Science (M.Sc.)
at the TUM School of Engineering and Design of the Technical University of Munich

Supervisor Prof. dr.ir. Daniel J. Rixen
Chair of Applied Mechanics

Advisor Francesco Trainotti M.Sc.,
Johannes Maierhofer M.Sc.
Chair of Applied Mechanics

Submitted by Jure Marinko

Submitted on May 30, 2022 in Garching

Acknowledgements

First of all, I would like to thank my thesis advisors Francesco Trainotti M.Sc. and Johannes Maierhofer M.Sc. for the numerous discussions, advice and their guidance throughout this project. I would also like to thank prof. dr. ir. Daniel J. Rixen for the support on that project at the Chair of Applied Mechanics.

I am grateful for the base knowledge learned at the University of Ljubljana, and I express my gratitude to the Technical University of Munich, where I was able to study in a professional and pleasant community. I would like to acknowledge the generous financial support of the Republic of Slovenia as a part of the scholarship Ad futura.

Finally, I would like to thank my parents, siblings and my wife for providing me with unfailing support during the study and for bringing a lot of joy into my life. This accomplishment would not have been possible without them. Thank you.

Jure Marinko

Abstract

Simulations are a popular and powerful tool in today's industry. Their main advantage is that they can perform experiments virtually instead of physically. As a result, the design and development process becomes cheaper and faster, which are two of the most important things in a business today.

As all areas of engineering evolve, so does the numerical complexity of the simulations. Especially with the usage of the complex materials such as rubber. Various techniques are under development to reduce the computational costs. In this thesis, we focus on two of the most promising. The reduction method called Proper Orthogonal Decomposition (POD) and the hyper-reduction method called Energy Conserving Sampling and Weighting (ECSW).

In the context of this work, we try to approach the industry. For this reason, the widely used commercial finite element software Abaqus is used. Since Abaqus is so far not able to perform all the steps necessary for a reduction, it is supplemented by Python code. The capabilities and limitations of Abaqus to perform POD and ECSW reductions are investigated in this thesis.

Zusammenfassung

Simulationen sind ein beliebtes und leistungsfähiges Hilfsmittel in der heutigen Industrie. Ihr Hauptvorteil besteht darin, dass sie Experimente virtuell und nicht physisch durchführen können. Dadurch wird der Design- und Entwicklungsprozess billiger und schneller, zwei der wichtigsten Dinge in der heutigen Geschäftswelt.

Da sich alle Bereiche der Technik weiterentwickeln, steigt auch die numerische Komplexität der Simulationen. Insbesondere bei der Verwendung von komplexen Materialien wie Gummi. Es werden verschiedene Techniken entwickelt, um die Rechenkosten zu reduzieren. In dieser Arbeit konzentrieren wir uns auf zwei der vielversprechendsten. Die Reduktionsmethode namens Proper Orthogonal Decomposition (POD) und die Hyper-Reduktionsmethode namens Energy Conserving Sampling and Weighting (ECSW).

Im Rahmen dieser Arbeit versuchen wir, uns der Industrie anzunähern. Aus diesem Grund wird das weit verbreitete kommerzielle Finite-Elemente-Programm Abaqus verwendet. Da Abaqus bisher nicht in der Lage ist, alle für eine Reduktion notwendigen Schritte durchzuführen, wird es durch Python-Code ergänzt. Die Möglichkeiten und Grenzen von Abaqus zur Durchführung von POD- und ECSW-Reduktionen werden in dieser Arbeit untersucht.

Contents

Chapter 1

Introduction

1.1 Motivation

Rubber is a commonly used material in today's industry. Due to its energy dissipating properties, it is particularly suitable as a damping element. Rubber material can be found, for example, in automotive or aerospace sectors as small connecting elements or even in buildings as viscoelastic layers. In most cases, it provides connections in larger multi-body systems. For the accurate prediction of the response of the multibody system, the connecting parts play a crucial role and should be modeled precisely.

Accurate modeling of rubber often means that large deformations must be considered because its stiffness is relatively low and the deformations are consequently large. In addition, the rubber material can exhibit nonlinear elastic behavior, and the response can be time (frequency), amplitude, and temperature dependent. Because of the mentioned nonlinear behaviour, numerical analysis involving rubber parts can become expensive. Reducing computational costs would enable more advanced and comprehensive simulations (e.g., optimizations) that are not currently affordable.

Various reduction techniques are under development to reduce the computational complexity of the finite element (FE) models. These techniques use different approaches to reduce the size of the problem and/or optimize the computational procedure. Currently, many reduction techniques have been tested at a scientific level but have not yet been performed in many industrial applications.

The aim of this work is to apply one of so called hyper-reduction techniques in the commercial FEM software Abaqus to tackle simulations involving a nonlinear viscoelastic rubber. The suitability of the reduction method for rubberlike materials and the suitability of Abaqus for hyper-reduction techniques are investigated.

1.2 Goal of the thesis

The objectives of the thesis can be summarized in three points:

1. Implement a strategy to reduce the computational cost of time-based simulation with complex nonlinear viscoelastic rubber behaviour.
2. Assess the suitability of the Python-Abaqus co-simulation for the implementation of a proper reduction strategy.
3. Assess the results of the reduced model compared to the original one in terms of accuracy and time.

1.3 Outline of content

The thesis is structured as follows: In Chapter 2, the theory of the Finite Element Method together with the theory for modeling of rubber-like materials is introduced. In addition, the full model used in this thesis is presented. Chapter 3 discusses the theory, implementation and results of the POD reduction. In Chapter 4 the theory, modeling and results of the ECSW hyper-reduction is presented. Finally, in chapter 5 some conclusions and ideas on further work are listed. Appendix A contains some Abaqus specific knowledge and explanations. Appendix B contains some additional results that were omitted from the main thesis.

Chapter 2

High fidelity modeling

In this chapter, the concept of high fidelity modeling of rubberlike parts is briefly introduced. At the beginning, some theoretical basics about finite element method and material models are explained. At the end of the theory, fundamental equations are summarized. After that, the Abaqus finite element model is presented, which is used throughout this thesis. At the end of the chapter, the first few eigenmodes of the model are shown to illustrate the fundamental dynamic properties of the model.

2.1 Theoretical background

2.1.1 Linear Finite Elements

Finite Element method (FEM) is a well-known and widely used method in today's engineering. The main benefit of the FEM is its flexibility. With this approach, any arbitrarily shaped domain can be discretized and solved. There is no need to derive an analytical solution for each single domain, if this is even possible. The main idea is to divide a domain into several smaller elements in which the solution is approximated with a known mathematical function (linear or quadratic are often chosen) corresponding to analytical equation of the problem. By dividing the domain in several smaller elements, the problem is discretized in terms of matrices and vectors. The evaluation is performed only at the integration points and then projected to the nodes. In the case of nonlinear material, some of these matrices and vectors become time or displacement dependent. It will be discussed in the subsections on nonlinearity in 2.1.2 and 2.1.3.

The theory of the FEM itself does not go into further details here. Only the most important aspects for the understanding of the thesis are pointed out. For a deeper understanding of the method, the reader is referred to one of the basic books explaining the FEM (e.g. [10]). According to [10], equation 9.11, the driving equation of the FEM mechanical dynamics is 2.1.

$$\mathbf{M}\ddot{\mathbf{x}}(t) + \boldsymbol{C}\dot{\boldsymbol{x}}(t) + \mathbf{K}\mathbf{x}(t) = \mathbf{F}_{ext}(t) \tag{2.1}$$

Here $\mathbf{M}$, $\boldsymbol{C}$, $\mathbf{K}$, $\ddot{\mathbf{x}}$, $\dot{\mathbf{x}}$, $\mathbf{x}$ and $\mathbf{F}_{ext}$ are the mass matrix, viscous-damping matrix, stiffness matrix, second and first time derivative of the displacement vector, displacement vector and vector of external forces, respectively.

In our application, materials do not exhibit purely viscous damping. For this reason, an equation could be simplified to 2.2. The damping term is omitted since no viscous damping is assumed.

$$\mathbf{M}\ddot{\mathbf{x}}(t) + \mathbf{K}\mathbf{x}(t) = \mathbf{F}_{ext}(t) \tag{2.2}$$

There are as many rows in a system of equations as there are degrees of freedom in a discretized model. Here, a limitation of the method becomes apparent. If the model consists of many degrees of freedom, the matrices become very large, which leads to high computational costs.

2.1.2 Nonlinearity 1 - Geometric nonlinearity

The first type of nonlinearity considered in this thesis is geometric. Starting from the physical world, in an essence any problem could be understood as geometrically nonlinear. However, some of them can be simplified to be modeled linearly if the displacements and rotations are relatively small compared to the dimensions of the object. In such a case, the result of the linear analysis is assessed as a good approximation of the nonlinear result. In our case, however, a nonlinear formulation is used because the displacements are expected to be relatively large.

The difference in the calculation procedure is that in the linear case all matrices and vectors are evaluated in a non-deformed configuration (once, at the beginning of the simulation), while when nonlinearity is taken into account, the balance is evaluated for the deformed configuration. This means that in each state of the solution a stiffness matrix and stiffness forces are updated according to the geometry change. This leads to higher computational costs.

A tangent matrix needed for iterative solver depends on current displacement field (equation 2.3).

$$\mathbf{K} = \mathbf{K}(\mathbf{x}) \tag{2.3}$$

Since $\boldsymbol{K}$ changes with $\boldsymbol{x}$, the stiffness forces ($\boldsymbol{f}_{stiff}(\boldsymbol{x})$) can no longer be evaluated with a multiplication of $\boldsymbol{K}\boldsymbol{x}$. They are extracted by applying the displacement field directly to the model. Equation 2.2 is reformulated for use in nonlinear geometric analysis - equation 2.4.

$$\mathbf{M}\ddot{\mathbf{x}}(t) + \boldsymbol{f}_{stiff}(\boldsymbol{x}(t)) = \mathbf{F}_{ext}(t) \tag{2.4}$$

2.1.3 Nonlinearity 2 - Material nonlinearity

Another source of nonlinearity is material nonlinearity. Rubber material has some peculiar properties, which will be presented in this subsection.

Hyperelasticity

Elasticity is an important material property that describes the ability of a given material to return to its original state after all loads have been removed. In other words, the material does not exhibit permanent deformation. Rubbers specifically are able to bear large elongations (see Ahmadi [1]). The relation between strain and stress is usually nonlinear but it

can be assumed that no energy is dissipated at a slow deformation rate. The elastic part of the rubber is therefore usually approximated with a hyperelastic model, which support this assumption. The typical hyperelastic stress-strain curve is shown in the figure 2.1.

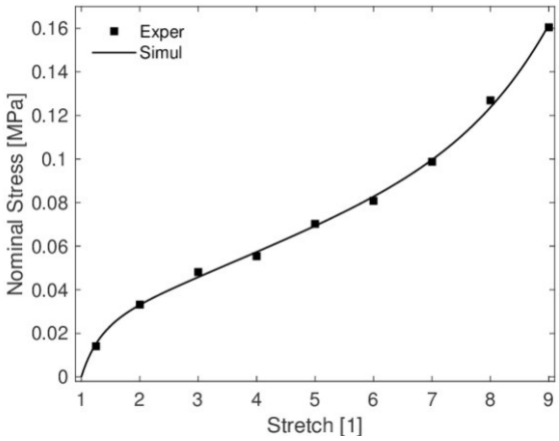


Figure (2.1) Typical hyperelastic stress-strain curve [11]

Hyperelasticity assumes that a relation between strain and stress is derived from a potential energy function, which means that the material does not dissipate energy. This assumption is valid for modeling of very slow loads, where viscosity effects are neglected. In this thesis, only Yeoh hyperelastic model is used. Equation 2.5 represents the definition of the energy for the Yeoh hyperelastic model. I_1 is the first invariant of the Cauchy-Green deformation tensor $\boldsymbol{C}$.

$$W = C_{10}(I_1 - 3) + C_{20}(I_1 - 3)^2 + C_{30}(I_1 - 3)^3 \tag{2.5}$$

The Cauchy stress is evaluated for a hyperelastic model as follows (see Austrell [4]):

$$\boldsymbol{\sigma} = 2J^{-1}\boldsymbol{F}\frac{\partial W}{\partial \boldsymbol{C}}\boldsymbol{F}^T - p\boldsymbol{I} \tag{2.6}$$

Then a stiffness matrix is directly proportional to the 2.7 missing some geometrical scale factor, which depends on the actual shape and dimension of the element.

$$\boldsymbol{K} \propto \frac{\partial \boldsymbol{\sigma}}{\partial \boldsymbol{\epsilon}} \tag{2.7}$$

In the equation 2.6, J is defined as a determinant of the deformation gradient tensor $\boldsymbol{F}$. p is the pressure defined as $p = \frac{1}{3}(\sigma_{11} + \sigma_{22} + \sigma_{33})$. For further explanation see [4]. In equation 2.7, $\boldsymbol{\epsilon}$ represents a strain tensor.

Finally, it can be summarized that $\boldsymbol{K}$ again depends on the displacement field. Simplified again following is valid:

$$\boldsymbol{K} = \boldsymbol{K}(\boldsymbol{x}) \tag{2.8}$$

Viscoelasticity

Properties of the rubber also depend on time. This dependence can be described in both the time and frequency domain equivalently. The time dependence is commonly modeled with a

spring and damper in a serial connection (figure 2.2). For a linear spring with stiffness E and a linear damper with viscosity η, such an element is called a Maxwell element. For simplicity, a 1D model is shown here, but the model could easily be extended to 3D applications.

Figure (2.2) Maxwell element

A general equation of the Maxwell element 2.9, where $\dot{\epsilon}$ is a strain rate and σ is a stress.

$$\dot{\epsilon} = \frac{\sigma}{\eta} + \frac{\dot{\sigma}}{E} \tag{2.9}$$

The Prony series is a material model consisting of several Maxwell elements and one elastic or hyperelastic element connected in parallel. The structure of the model is shown in the figure 2.3. The stress is given by the equation 2.10, where the stress in each branch (σ_i) is evaluated separately.

$$\sigma = \sum_{i=0}^{n} \sigma_i \tag{2.10}$$

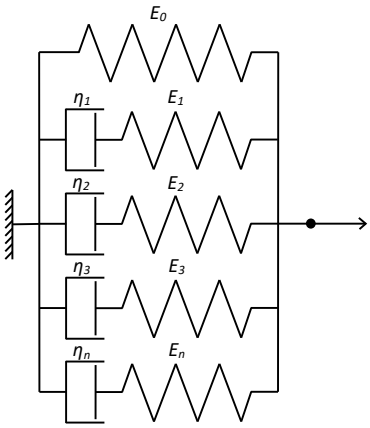


Figure (2.3) Prony series

It can be seen that as the time dependent properties are captured, the stiffness matrix also becomes time dependent. Even at fixed strain, the stiffness is changing with time.

$$\mathbf{K} = \mathbf{K}(t) \tag{2.11}$$

Abaqus definition of viscoelastic material

In Abaqus, the same constitutive model is written in terms of other variables. Instead of η_i (viscosity), the relaxation time, τ_i, is used. A relation is:

$$\tau_i = \frac{\eta_i}{E_i} \tag{2.12}$$

Instead of E_i, g_i is used, which is a ratio between the stiffness of the spring in branch i and the total stiffness of the model E_{tot}.

$$E_{tot} = E_0 + E_1 + E_2 + E_3 + ... + E_n \tag{2.13}$$

$$g_i = \frac{E_i}{E_{tot}} \tag{2.14}$$

Thus, the model definition in Abaqus is defined by the parameters τ_i and g_i (figure 2.4).

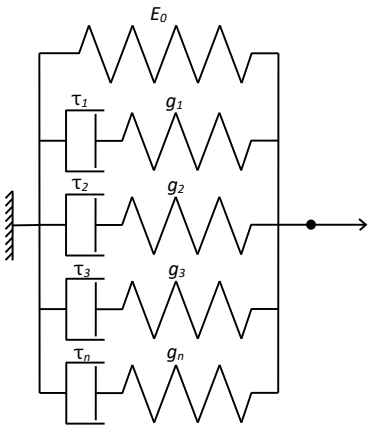


Figure (2.4) Prony series definition with Abaqus' parameters

2.1.4 Final equation of the problem

Equation 2.4 could be rewritten with additional dependence of the stiffness forces on time. A full nonlinear formulation is written in 2.15, where stiffness matrix changes for every different displacement and time state (2.16).

$$\mathbf{M}\ddot{\mathbf{x}}(t) + \boldsymbol{f}_{stiff}(\boldsymbol{x}, t) = \mathbf{F}_{ext}(t) \tag{2.15}$$

$$\boldsymbol{K} = \boldsymbol{K}(\boldsymbol{x}, t) \tag{2.16}$$

2.2 Full order reference system

Here, the basic properties of the model used in this thesis are presented. This is a reference model. In following chapters, the solutions of the reduced models will be compared with those generated with the reference model.

2.2.1 Geometry

The model is a simple cantilever beam depicted in 2.5. The dimensions a, b and c are denoted as length, thickness and width, respectively. The beam is rigidly attached on the left-hand side and the external force $\boldsymbol{F}_{ext}$ is plotted symbolically only. The force is not

always applied at the corner. Different load cases with loads at different locations will be defined later.

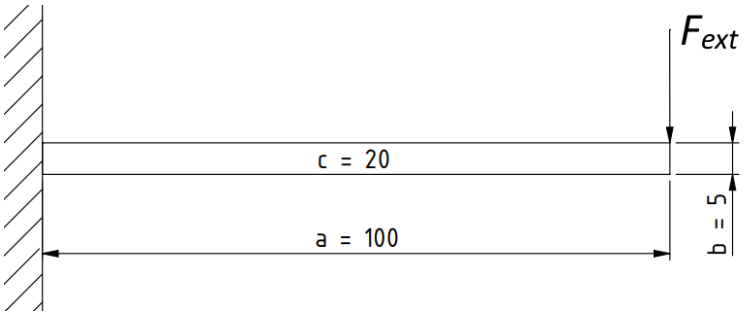


Figure (2.5) Sketch of the beam

2.2.2 Material

The material used is an EPDM 50 (where 50 represents the Shore Hardness A). The material data was provided directly by the Chair of Applied Mechanics, where the behavior of the rubberlike materials has already been investigated. The material card consists of the following data:

- Density: $1116\frac{kg}{m^3} = 1.116e-9\frac{ton}{mm^3}$
  For Abaqus, we have to provide a consistent system of units. For this reason, the definition is given in $\frac{ton}{mm^3}$.
- Hyperelastic: A reduced polynomial model of the 3rd order is used, which is another name for the Yeoh model. The required constants are C_{10}, C_{20}, C_{30}, which represent the nonlinear elasticity of the material and D_1, D_2, D_3, which represent the compressibility of the material.

C_{10}	C_{20}	C_{30}	D_1	D_2	D_3
0.40775	−0.277449	0.266309	0.024876	0	0

- Viscoelastic: A material is built up of 4 branches of the Prony series. The data are listed in the table below. To understand the material definition, see the 2.1.3 section.

branch	g_i	τ_i
1	0.4900	1.43061
2	0.1010	0.03079
3	0.1047	0.00482
4	0.1916	0.00058

Additional remark on combining of hyperelasticity and viscoelasticity in Abaqus. The rheological model remains the same as in the figure 2.4. The only difference is that the definition of the base spring E_0 becomes nonlinear and is driven by the hyperelastic law, in our case defined by C_{10}, C_{20} and C_{30}. Consequently, all branches become nonlinear since their stiffness is defined as a ratio related to the base branch.

2.2.3 Mesh

Linear hexahedral elements without reduced integration are used in the model (called C3D8 in Abaqus). The total number of elements is 80, and the total number of nodes is 210. A

mesh is visualized in the figure below. The nodes that are used wherever in the thesis for excitation are labeled in red. On the contrary, the nodes that are used to "record" the node path and plotting it, are labeled in yellow.

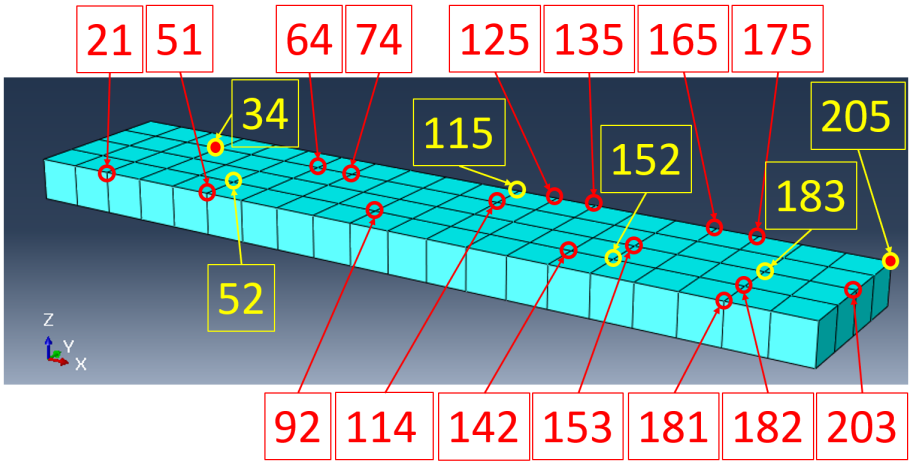


Figure (2.6) Mesh with labeled characteristic nodes.

2.2.4 Time integration

Since it is tried to prove the behavior of the time dependent material, including inertia forces, the steps in Abaqus are of the dynamic type. Abaqus uses Newmark time discretization scheme by default (the hht algorithm), but here Backward Euler scheme is used, as explained in the following chapter. In order to obtain results as comparable as possible with the reduced models, simulations in Abaqus are also performed using Backward Euler (BWE). BWE is enforced by manual overwriting the step command in the input file. Instead of just "*Dynamic", "*Dynamic, TIME INTEGRATOR=BWE" is used.

The duration of the step is always one cycle ($t_{end} = \frac{1}{f}$). With f being a frequency of excitation. The number of increments is fixed at 30 for all validation cases. The number of increments for the training simulations is also fixed, here 100 increments are used. The time step is calculated according to the time of simulation t_{end} and the number of increments desired. A different time step is chosen for different frequencies.

2.2.5 Boundary conditions

A beam has all displacements in all directions on the left-hand side set to 0. It is shown in the figure below with the orange symbols.

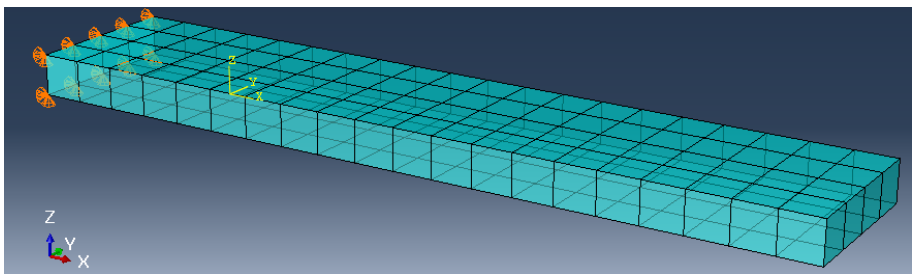


Figure (2.7) Boundary conditions of the model

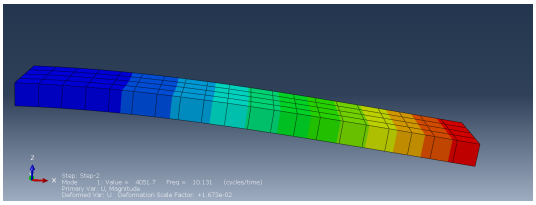
2.2.6 Load cases

A load is applied as sin on the defined nodes always in z-direction (vertical). The load cases differ by the magnitude X, frequency f, and the set of nodes ψ on which the load is applied. The vector $\boldsymbol{F}_{ext}$ is formed as a zero vector everywhere except at the locations of the z-component of the node from the set ψ.

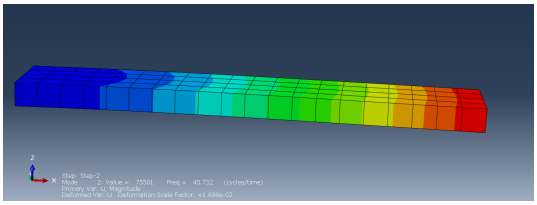
$$\boldsymbol{F}_{ext} = \begin{cases} -X \cdot sin(2\pi f \cdot t) & \text{if node in } \psi\text{; for z component} \\ 0 & \text{otherwise} \end{cases} \tag{2.17}$$

2.3 Eigenmodes of the full model

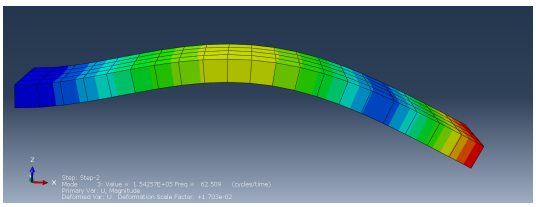
Here the eigenmodes of the full model are plotted. Just to get a rough feeling of the dynamics of the model.



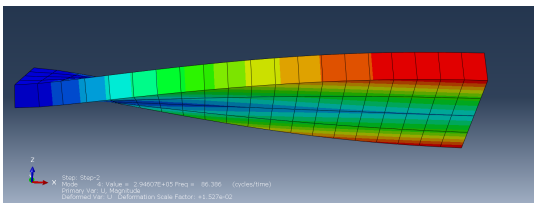
(a) Mode 1



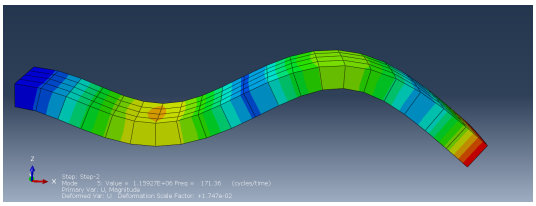
(b) Mode 2



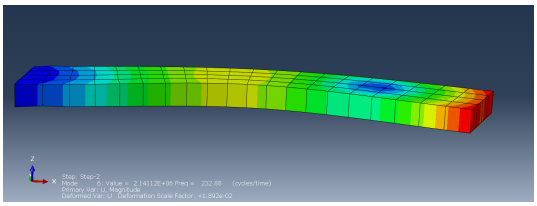
(c) Mode 3



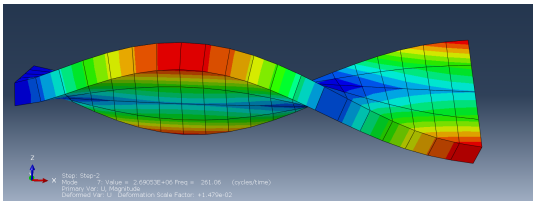
(d) Mode 4



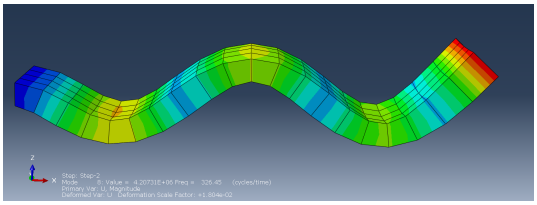
(e) Mode 5



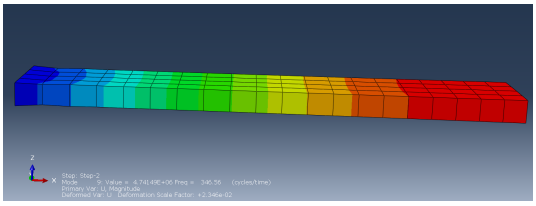
(f) Mode 6



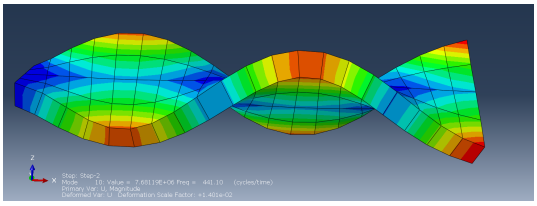
(g) Mode 7



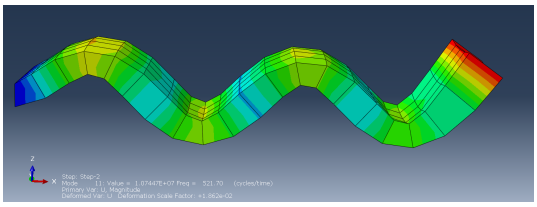
(h) Mode 8



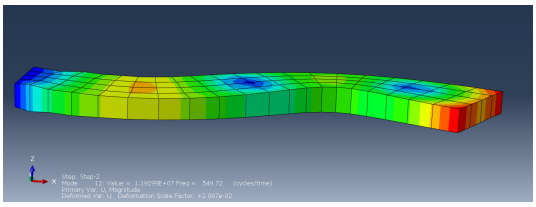
(i) Mode 9



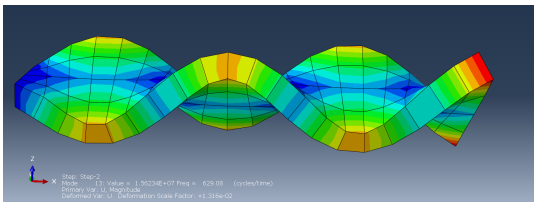
(j) Mode 10



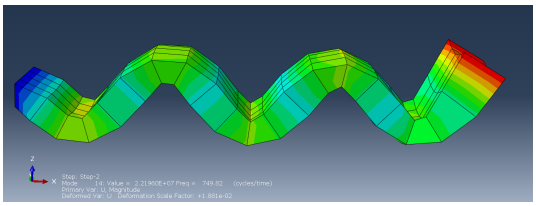
(k) Mode 11



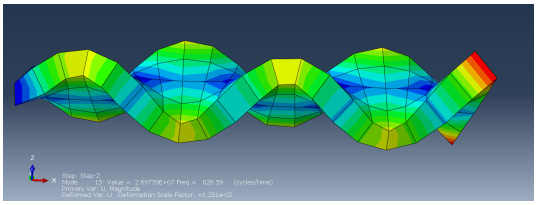
(l) Mode 12



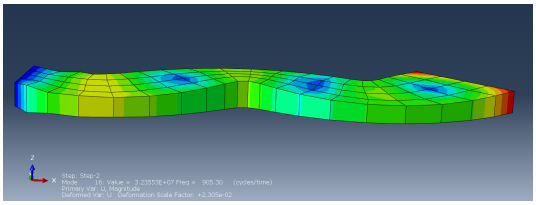
(m) Mode 13



(n) Mode 14



(o) Mode 15



(p) Mode 16

Figure (2.8) The eigenmodes of the full model

Chapter 3

Projection based reduction

In this chapter, all the content related to projection based reduction and in particular Proper orthogonal decomposition (POD) are presented. First some theoretical background is introduced, then the implementation in Python-Abaqus co-simulation is explained and finally the results are shown along with the discussion.

3.1 Theoretical background

3.1.1 General

Projection based reduction methods have in some sense the same underlying idea as the finite element method (Ritz or Galerkin method). The finite element method aims to reduce the number of degrees of freedom from an infinite to a finite number, assuming some shape functions known in advance. The projection based reduction methods reduce the number of degrees of freedom from a large finite number, to a smaller finite number. Reduction is achieved with new assumed functions that span the entire domain. The final deformation state is thus assembled from several shape functions that are premultiplied by participation factors.

A theory can be understood mathematically as a projection onto a new space consisting of fewer dimensions. The basis vectors of the reduced space are rows in the matrix $\boldsymbol{V}$. A new set of degrees of freedom $\boldsymbol{x}_{red}$ is introduced. The following is valid:

$$\boldsymbol{x} = \boldsymbol{V}\boldsymbol{x}_{red} \tag{3.1}$$

Then, the original linear equation 2.2 is written as:

$$\boldsymbol{M}\boldsymbol{V}\ddot{\boldsymbol{x}}_{red}(t) + \boldsymbol{K}\boldsymbol{V}\boldsymbol{x}_{red}(t) = \boldsymbol{F}_{ext}(t) + \boldsymbol{r} \tag{3.2}$$

Equation 3.2 represents an over-determined system to be solved. According to [2], the residual $\boldsymbol{r}$ could be interpreted as a remainder that is inevitable, since a reduced model is in general not able to express an exact solution of the full model. As written in [2], one requires that residual forces are zero at least in the possible directions of motion (which are defined in $\boldsymbol{V}$). Mathematically it is expressed as:

$$\boldsymbol{V}^T\boldsymbol{r} = \boldsymbol{0} \tag{3.3}$$

Then the entire equation 3.2 is premultiplied by a $\boldsymbol{V}^T$ from the left-hand side. The reduced equation for a linear problem reads:

$$\boldsymbol{V}^T\boldsymbol{M}\boldsymbol{V}\ddot{\boldsymbol{x}}_{red}(t) + \boldsymbol{V}^T\boldsymbol{K}\boldsymbol{V}\boldsymbol{x}_{red}(t) = \boldsymbol{V}^T\boldsymbol{F}_{ext}(t) \tag{3.4}$$

For a nonlinear problem, the following equation is valid:

$$\boldsymbol{V}^T\boldsymbol{M}\boldsymbol{V}\ddot{\boldsymbol{x}}_{red}(t) + \boldsymbol{V}^T\boldsymbol{f}_{stiff}(\boldsymbol{x},t) = \boldsymbol{V}^T\boldsymbol{F}_{ext}(t) \tag{3.5}$$

The underlying theory explained above is valid for all projection based reduction methods. The methods differ only in the way the reduction basis $\boldsymbol{V}$ is constructed. Fundamentally, two different types of methods are known. The methods of the first type are simulation-free methods, while those of the second type are simulation based methods. The first obtain the basis only from the properties of the system (e.g. mass distribution, stiffness distribution), while the second ones evaluate the reduction matrix using some data from the training simulations. Performing training simulations automatically means an initial computational effort that needs to be done on the full model.

Simulation-free methods

One of the widely known simulation-free concepts is modal truncation. The idea originates from Lord Rayleigh in 1894 [15]. It transforms the model from the spatial coordinates to the modal ones and truncates the space there. There are other methods, also developed very early, some of which are still state of the art. For example; Guyan reduction method [8], Craig-Bampton method [7] and methods of MacNeal [13] or Rubin [16]. However, these methods are efficiently applied only to the linear systems. For the nonlinear systems it becomes much more demanding. About simulation-free methods applied to the nonlinear systems can be read in [17]. They are based on linearization of the system (similar to the methods for linear systems), with additional capturing of the nonlinearity via information obtained with perturbation techniques. However, as explained in [17] it is difficult to obtain invariant and meaningful physical properties for an arbitrary nonlinear system.

Simulation-based methods

The second class of strategies is based on training simulations. The same idea of the model order reduction as before is used. Only the reduction matrix is evaluated in a different way. In this case, the reduction space is not calculated purely based on mathematical properties of the system. Rather, it is computed from snapshots of the training simulation. A limitation of this approach is that the deformation states in the training simulation should correspond to what the model should be capable of later. Such a model loses a part of generality.

One of the most commonly used methods of this type is Proper Orthogonal Decomposition (POD), which according to [5] was developed independently by several authors. Two of them are Karhunen [9] and Loève [12], after whom the method is also called Karhunen-Loève decomposition. In this work it will be called with the abbreviation POD.

3.1.2 Proper Orthogonal Decomposition (POD)

As explained above, the POD method belongs to the simulation-based methods. Here, the reduction basis consists of the most important singular modes computed with Singular value decomposition (SVD). This is a statistical tool that finds the most frequently participated deformation shapes from the training simulation snapshots, which are stored in a matrix $\boldsymbol{S}$. SVD assigns left singular vectors to the matrix $\boldsymbol{A}$ and right singular vectors to the matrix $\boldsymbol{B}$.

Singular values are stored in the diagonal matrix $\mathbf{\Sigma}$. They are sorted according to their importance, which is expressed by the magnitude of the corresponding singular values. The higher singular value, more important the mode is.

$$\boldsymbol{S} = \boldsymbol{A}\mathbf{\Sigma}\boldsymbol{B}^T \tag{3.6}$$

Some left singular vectors from the matrix $\boldsymbol{A}$ (the number is specified by the user) are selected to build a reduction basis $\boldsymbol{V}$ - a space in which the object can deform.

Here some basic definitions are listed for further derivation:

- $1 < d < ts$ with ts being the overall number of training simulations.
- x_{ij} is the position coordinate with i as the index of the snapshot and j as the index of the degree of freedom (dof).
- $1 < i < s$ with s being the number of the snapshots in one training simulation.
- $1 < j < N$ with N being the number of the dof-s in original model.
- $\boldsymbol{A}$ is a matrix of the left singular vectors of the matrix $\boldsymbol{S}_d$. The vectors are orthogonal to each other and they are normalized to the length 1.
- $\boldsymbol{B}$ is a matrix of the right singular vectors of the matrix $\boldsymbol{S}_d$. The vectors are orthogonal to each other and they are normalized to the length 1.
- $\mathbf{\Sigma}$ is a diagonal matrix of the singular values of the matrix $\boldsymbol{S}_d$.
- n is a user defined number of the left singular vectors used in the reduction matrix $\boldsymbol{V}$.
- θ_d is a scaling factor calculated for each training simulation. The explanation of it follows below.

$$\widetilde{\boldsymbol{S}}_d = \begin{bmatrix} x_{11} & \dots & x_{s1} \\ \vdots & \ddots & \vdots \\ x_{1N} & \dots & x_{sN} \end{bmatrix} \tag{3.7}$$

A matrix $\widetilde{\boldsymbol{S}}_d$ has a shape $N \times s$. It is directly assembled from snapshot vectors. We could use this original matrix directly for further calculation. However, it is recommended to normalize the datasets as this improves the results. Since the representative shape vectors are chosen statistically and only based on the snapshots, the simulation snapshots should be of approximately the same order of magnitude to have an equivalent influence. This can be assured by scaling the entire training simulation displacement field by an appropriate factor to compensate the differences in magnitude between the training simulations. This basically means that the absolute largest value of the training simulation is set to a constant value ζ for all the training simulations. The magnitude of ζ does not affect the solution in theory and could be arbitrary chosen (e.g. 100 in our case). In practice, it can affect numerical conditioning of the matrix a very small or very large value is chosen. A scaling factor is calculated as follows:

$$S_{d,max} = max_{abs}\left(\begin{bmatrix} x_{11} & \dots & x_{s1} \\ \vdots & \ddots & \vdots \\ x_{1N} & \dots & x_{sN} \end{bmatrix}\right) \tag{3.8}$$

$$\theta_d = \frac{\zeta}{S_{d,max}} \tag{3.9}$$

$$\boldsymbol{S}_d = \begin{bmatrix} x_{11} & \dots & x_{s1} \\ \vdots & \ddots & \vdots \\ x_{1N} & \dots & x_{sN} \end{bmatrix} \star \theta_d \tag{3.10}$$

Once all $\boldsymbol{S}_d$ matrices are scaled, they are combined in one large matrix as follows.

$$\boldsymbol{S} = \left[\left[\begin{array}{c} \\ \boldsymbol{S}_1 \\ \\ \end{array} \right] \dots \left[\begin{array}{c} \\ \boldsymbol{S}_d \\ \\ \end{array} \right] \dots \left[\begin{array}{c} \\ \boldsymbol{S}_{ts} \\ \\ \end{array} \right] \right] \tag{3.11}$$

SVD is performed for the matrix $\boldsymbol{S}$:

$$\boldsymbol{S} = \boldsymbol{A}\boldsymbol{\Sigma}\boldsymbol{B}^T \tag{3.12}$$

For the reduction matrix V, a user defined number n of columns is taken from the left singular matrix $\boldsymbol{A}$. n becomes the dimension of the reduced model. Note that a scalar value A is an entry of the matrix $\boldsymbol{A}$.

$$\boldsymbol{V}^T = \begin{bmatrix} A_{11} & \dots & A_{1n} \\ \vdots & \ddots & \vdots \\ A_{N1} & \dots & A_{Nn} \end{bmatrix} \tag{3.13}$$

Alternative SVD approach
Alternatively, the SVD can be performed on a subset of the snapshots $\boldsymbol{S}_d$ of each training simulation separately. From each left singular matrix $\boldsymbol{A}_d$, only some important vectors are transmitted to the general matrix $\boldsymbol{S}$. With doing so, the matrix $\boldsymbol{S}$ is reduced in size by omitting the less important basis vectors. Then, the SVD is applied to the matrix $\boldsymbol{S}$ to eliminate all redundant mode shapes and build an orthogonal basis $\boldsymbol{A}$. An user defined number of the columns of the matrix $\boldsymbol{A}$ are then used in the reduction basis $\boldsymbol{V}$. This could have a positive effect on the computational cost for large models with many training simulations.

3.1.3 Solving procedure

Once we have a reduction matrix $\boldsymbol{V}$, the calculation procedure for a reduced model must be developed. The equation 3.4 is not suitable to be run in Abaqus, since Abaqus does not support calculations with user defined reduction matrix. This is one of the main reasons why the calculation will be run in a Python-Abaqus co-simulation.

A derivation will start from the linear equation 3.4. Then the Backward-Euler time discretization, the nonlinearity and the Newton-Raphson root finding algorithm will be added.

Linear first order differential formulation

The starting point is a linear equation of the reduced problem - equation 3.4. It is a second order differential equation, but it can be rewritten into a first order differential equation to become more convenient for the solving. It is done with substitution of variables. We introduce a new vector variable $\boldsymbol{q}$ as a concatenation of $\dot{\boldsymbol{x}}_{red}$ and $\boldsymbol{x}_{red}$:

$$\boldsymbol{q} = \begin{bmatrix} \dot{\boldsymbol{x}}_{red} \\ \boldsymbol{x}_{red} \end{bmatrix} \tag{3.14}$$

After substitution we obtain the equation 3.15. $\boldsymbol{I}$ is an identity matrix of the appropriate size. The size of the problem is $2n \times 2n$.

$$\begin{bmatrix} \boldsymbol{V}^T\boldsymbol{M}\boldsymbol{V} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \dot{\boldsymbol{q}}(t) + \begin{bmatrix} \boldsymbol{0} & \boldsymbol{V}^T\boldsymbol{K}\boldsymbol{V} \\ -\boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \boldsymbol{q}(t) = \begin{bmatrix} \boldsymbol{V}^T\boldsymbol{F}_{ext}(t) \\ \boldsymbol{0} \end{bmatrix} \tag{3.15}$$

Backward Euler time discretization

To solve the problem presented above, time must be discretized. Here this is done with a Backward Euler scheme, where the derivative is approximated as follows:

$$\dot{\boldsymbol{q}}_{n+1} = \frac{\boldsymbol{q}_{n+1} - \boldsymbol{q}_n}{\Delta t_{inc}} \tag{3.16}$$

Δt_{inc} is a time step between two increments. The subscript n denotes a value at increment n. n stands for the previous time increment which is already solved, while $n+1$ denotes the current one, which is being solved. For the sake of simplicity, we are using notation $\boldsymbol{q}_{n+1}$, which corresponds to $\boldsymbol{q}(t_{n+1})$.

A new variable $\Delta\boldsymbol{q}_{n+1}$ is introduced as a change of $\boldsymbol{q}$ between increments $n+1$ and n. The relation is defined in 3.17.

$$\Delta\boldsymbol{q}_{n+1} = \boldsymbol{q}_{n+1} - \boldsymbol{q}_n \tag{3.17}$$

A time-discretized equation is rewritten to:

$$\begin{bmatrix} \boldsymbol{V}^T\boldsymbol{M}\boldsymbol{V} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \frac{\Delta\boldsymbol{q}_{n+1}}{\Delta t_{inc}} + \begin{bmatrix} \boldsymbol{0} & \boldsymbol{V}^T\boldsymbol{K}\boldsymbol{V} \\ -\boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \boldsymbol{q}_{n+1} = \begin{bmatrix} \boldsymbol{V}^T\boldsymbol{F}_{ext,n+1} \\ \boldsymbol{0} \end{bmatrix} \tag{3.18}$$

Nonlinear formulation

As discussed in 2.1.2 and 2.1.3, a stiffness matrix becomes displacement and time dependent when nonlinear effects are considered ($\boldsymbol{K} = \boldsymbol{K}(\boldsymbol{x}, t)$). With such an assumption, a linear second term in the equation 3.18 should be replaced by a term representing nonlinear contributions. A vector is defined in 3.19. Variable $\boldsymbol{v}$ represents a velocity in the reduced space.

$$\begin{bmatrix} \boldsymbol{V}^T\boldsymbol{f}_{stiff}(\boldsymbol{x}_{n+1}, t_{n+1}) \\ \boldsymbol{v}_{n+1} \end{bmatrix} \tag{3.19}$$

A nonlinear formulation of the problem is obtained (equation 3.20).

$$\begin{bmatrix} \boldsymbol{V}^T\boldsymbol{M}\boldsymbol{V} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \frac{\Delta\boldsymbol{q}_{n+1}}{\Delta t_{inc}} + \begin{bmatrix} \boldsymbol{V}^T\boldsymbol{f}_{stiff}(\boldsymbol{x}_{n+1}, t_{n+1}) \\ \boldsymbol{v}_{n+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{V}^T\boldsymbol{F}_{ext,n+1} \\ \boldsymbol{0} \end{bmatrix} \tag{3.20}$$

Newton-Raphson method

Because of the nonlinear problem, a root finding method must be implemented. Here, a Newton-Raphson method is used. It is an iterative solver which solves each increment $_{n+1}$ in several smaller iterations ^{i}. The label ^{i} represents the solved iteration, and $^{i+1}$ the iteration being solved. The iterations follow each other in ascending order, with the forerunner of $^{i=0}_{n+1}$ being $^{i=last}_{n}$.

Stiffness term

In order to formulate a Newton-Raphson algorithm, the second term of the equation 3.20 is rewritten as 3.21. The first term here are values calculated in the last solved iteration ^{i}. The second term is a linearized correction that will be calculated for the iteration $^{i+1}$.

$$\begin{bmatrix} \boldsymbol{V}^T \boldsymbol{f}_{stiff}(\boldsymbol{x}_{n+1}, t_{n+1}) \\ \boldsymbol{v}_{n+1} \end{bmatrix} \rightarrow \begin{bmatrix} \boldsymbol{V}^T \boldsymbol{f}^i_{stiff,n+1} \\ \boldsymbol{v}^i_{n+1} \end{bmatrix} + \begin{bmatrix} \boldsymbol{0} & \boldsymbol{V}^T \boldsymbol{K}^i_{n+1} \boldsymbol{V} \\ -\boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \Delta \boldsymbol{q}^{i+1}_{n+1} \tag{3.21}$$

In an initial, undeformed state, the vector 3.22 is a zero vector. Later on, the velocity vector $\boldsymbol{v}$ is added at each iteration as shown in 3.23, while the vector of stiffness forces $\boldsymbol{f}^{i+1}_{stiff,n+1}$ is updated by applying the displacement field $\boldsymbol{x}^{i+1}_{n+1}$ to the Abaqus model at every iteration (equation 3.24). In the same way, a stiffness matrix $\boldsymbol{K}^{i+1}_{n+1}$ is updated for every iteration by applying of the displacement field in Abaqus (equation 3.25). Some specific details about the Abaqus implementation are given in the appendix A.

$$\begin{bmatrix} \boldsymbol{f}^{i=0}_{stiff,n=0} \\ \boldsymbol{v}^{i=0}_{n=0} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix} \tag{3.22}$$

$$\boldsymbol{v}^{i+1}_{n+1} = \boldsymbol{v}^{i}_{n+1} + \begin{bmatrix} -\boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \Delta \boldsymbol{q}^{i+1}_{n+1} \tag{3.23}$$

$$\boldsymbol{f}^{i+1}_{stiff,n+1} = \boldsymbol{f}_{stiff}(\boldsymbol{x}^{i+1}_{n+1}, t_{n+1}) \tag{3.24}$$

$$\boldsymbol{K}^{i+1}_{n+1} = \boldsymbol{K}(\boldsymbol{x}^{i+1}_{n+1}, t_{n+1}) \tag{3.25}$$

Inertia term

When transforming from the increment to the iteration formulation, one more thing has to be changed. Referring to equation 3.20, $\Delta \boldsymbol{q}_{n+1}$ in the first term (inertia term) cannot be simply replaced by $\Delta \boldsymbol{q}^{i+1}_{n+1}$. In that case some inertia forces would be missed, because Δt_{inc} refers to an increment time (not of the iteration). Instead, the first term $\Delta \boldsymbol{q}_{n+1}$ is replaced by $d\boldsymbol{q}^{i} + \Delta \boldsymbol{q}^{i+1}_{n+1}$. $d\boldsymbol{q}^{i}$ is a new variable that is set to zero at the beginning of each increment and then accumulates the changes of $\Delta \boldsymbol{q}^{i+1}_{n+1}$ within one increment. At the beginning of every increment:

$$d\boldsymbol{q}^{i} = \boldsymbol{0} \tag{3.26}$$

After each iteration:

$$d\boldsymbol{q}^{i+1} = d\boldsymbol{q}^{i} + \Delta \boldsymbol{q}^{i+1}_{n+1} \tag{3.27}$$

The inertia term could then be rewritten as 3.28. After the second equality, $\boldsymbol{f}_{inert}$ introduces another simplification. It is a force vector correlated with $d\boldsymbol{q}$. It is $\boldsymbol{0}$ at the beginning of each

increment when $d\boldsymbol{q}$ is also zero. Later on, it accumulates $\delta\boldsymbol{q}_{n+1}^{i+1}$ contributions of each iteration as shown in 3.29.

$$\begin{bmatrix} \boldsymbol{V}^T\boldsymbol{M}\boldsymbol{V} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \frac{\Delta\boldsymbol{q}_{n+1}}{\Delta t_{inc}} = \begin{bmatrix} \boldsymbol{V}^T\boldsymbol{M}\boldsymbol{V} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \frac{d\boldsymbol{q}^i}{\Delta t_{inc}} + \begin{bmatrix} \boldsymbol{V}^T\boldsymbol{M}\boldsymbol{V} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \frac{\Delta\boldsymbol{q}_{n+1}^{i+1}}{\Delta t_{inc}} = \boldsymbol{f}_{inert}^i + \frac{1}{\Delta t_{inc}} \begin{bmatrix} \boldsymbol{V}^T\boldsymbol{M}\boldsymbol{V} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \Delta\boldsymbol{q}_{n+1}^{i+1} \tag{3.28}$$

$$\boldsymbol{f}_{inert}^{i+1} = \boldsymbol{f}_{inert}^i + \frac{1}{\Delta t_{inc}} \begin{bmatrix} \boldsymbol{V}^T\boldsymbol{M}\boldsymbol{V} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \Delta\boldsymbol{q}_{n+1}^{i+1} \tag{3.29}$$

Final formulation

Considering all explained, the equation 3.30 can be rewritten as follows:

$$\left(\frac{1}{\Delta t_{inc}} \begin{bmatrix} \boldsymbol{V}^T\boldsymbol{M}\boldsymbol{V} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} + \begin{bmatrix} \boldsymbol{0} & \boldsymbol{V}^T\boldsymbol{K}_{n+1}^i\boldsymbol{V} \\ -\boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \right) \Delta\boldsymbol{q}_{n+1}^{i+1} = \begin{bmatrix} \boldsymbol{V}^T\boldsymbol{F}_{ext,n+1} \\ \boldsymbol{0} \end{bmatrix} - \begin{bmatrix} \boldsymbol{V}^T\boldsymbol{f}_{stiff}(\boldsymbol{x}_{n+1}^i) \\ \boldsymbol{v}_{n+1}^i \end{bmatrix} - \begin{bmatrix} \boldsymbol{f}_{inert}^i \end{bmatrix} \tag{3.30}$$

In a more general sense, the right-hand side of the equation 3.30 can be understood as the residual $\boldsymbol{r}$ and the left-hand side as the preconditioner $\boldsymbol{X}$. Both depend on the last known displacement field $\boldsymbol{x}_{n+1}^i$ and an incremental time t_{n+1}.

$$\boldsymbol{r}_{n+1}^i = \begin{bmatrix} \boldsymbol{V}^T\boldsymbol{F}_{ext,n+1} \\ \boldsymbol{0} \end{bmatrix} - \begin{bmatrix} \boldsymbol{V}^T\boldsymbol{f}_{stiff}(\boldsymbol{x}_{n+1}^i) \\ \boldsymbol{v}_{n+1}^i \end{bmatrix} - \begin{bmatrix} \boldsymbol{f}_{inert}^i \end{bmatrix} \tag{3.31}$$

$$\boldsymbol{X}_{n+1}^i = \left(\frac{1}{\Delta t_{inc}} \begin{bmatrix} \boldsymbol{V}^T\boldsymbol{M}\boldsymbol{V} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} + \begin{bmatrix} \boldsymbol{0} & \boldsymbol{V}^T\boldsymbol{K}_{n+1}^i\boldsymbol{V} \\ -\boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \right) \tag{3.32}$$

Finally, the equation 3.33 is obtained.

$$\boldsymbol{X}_{n+1}^i \Delta\boldsymbol{q}_{n+1}^{i+1} = \boldsymbol{r}_{n+1}^i \tag{3.33}$$

3.1.4 Co-simulation approach

Here is a graphical representation of the procedure derived above. The point is to clearly summarize the structure of the code and the Python-Abaqus interactions. The calculation of the reduction basis $\boldsymbol{V}$ is depicted in the figure 3.1, and the co-simulation procedure for calculation with a reduced model is shown in the figure 3.2. The mass matrix $\boldsymbol{M}$ is assumed to be displacement independent and is calculated once at the beginning. $\boldsymbol{K}$ and $\boldsymbol{f}_{stiff}$ are calculated in each iteration based on displacement field $\boldsymbol{x}$ and time moment t received from Python. Equations for $\boldsymbol{X}$ and $\boldsymbol{r}$ are derived in 3.32 and 3.31. Detailed pseudo-code is written in the following section (3.2).

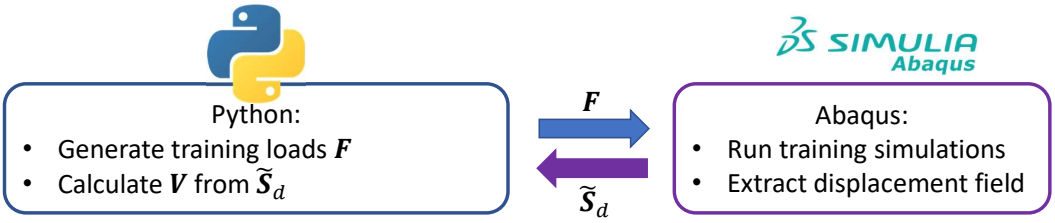


Figure (3.1) Calculation of the reduction basis $\boldsymbol{V}$

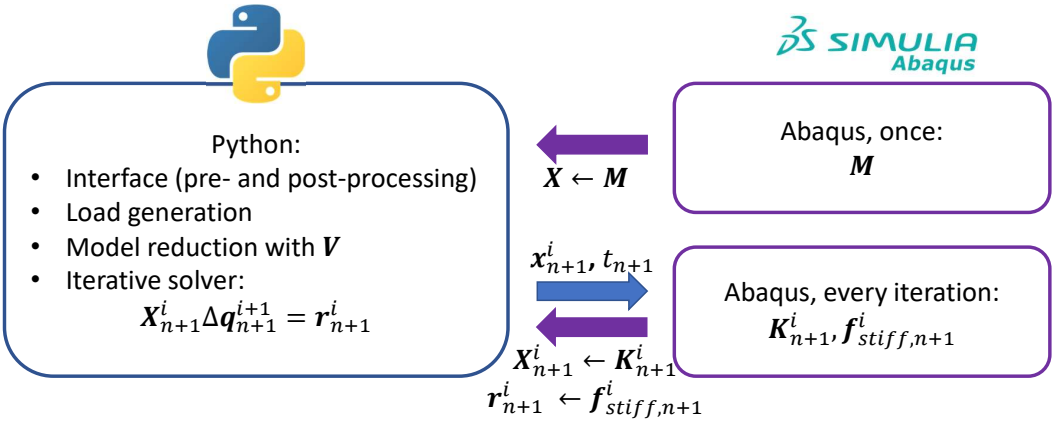


Figure (3.2) Co-simulation of the reduced model

3.2 Implementation

3.2.1 Calculate reduction basis

A rough overview of the procedure for calculation of the reduction basis is done in figure 3.1, and the theory is explained in 3.1.2. Here, additional details that apply for our implementation case are described and a detailed pseudo-code is shown.

In general, several different training simulations are performed with different values for amplitude, frequency and excitation node. The reason for running of several different simulations is that we want a reduction basis that could captures dynamics over a wide range of frequencies and also capture viscoelasticity (different frequency training sets) and also nonlinear behaviour such as hyperelasticity or geometrical nonlinearity (different amplitude training sets). The training simulations are of the dynamic type in our case, since time and inertia are important factors that we want to capture. The loads are defined according to definition described in 2.2.6. Each training simulation is forced to be solved with 100 equal long increments. Each time increment corresponds to one training snapshot. Thus, the number of the snapshots per training simulation is $s = 100$. The second of the methods presented in 3.1.2 is used, namely One big SVD. In our case, where the model is small it is easier to collect all snapshots in one big matrix and perform SVD once, instead of performing SVD for each training simulation. For the normalization procedure, the factor ζ is set to 100. As it was explained in the theory, this is an arbitrary chosen value. The main workflow runs in Python. The steps that require interaction with Abaqus are marked with a comment "// in Abaqus". Some details of the Abaqus implementation can be found in the appendix A.

Algorithm 1: POD basis calculation

Input: *Ab_model* ... Abaqus model with defined mesh and material properties
LCs ... list of load cases. Load case is a dictionary with defined:
– f ... frequency of the load
– X ... amplitude of the load
– ψ ... set of nodes on which the load is applied
n ... number of basis vectors used in reduction basis

Output: Reduction basis $\boldsymbol{V}$

initialize $\boldsymbol{S}$; // Empty matrix for collecting snapshots.
$ts \leftarrow \text{length}(LCs)$;
for $d \leftarrow 1$ **to** ts **do**
  $LC \leftarrow LCs(\text{d})$;
  Set up *Ab_model* with load LC ; // in Abaqus
  Run *Ab_model* simulation in Abaqus ; // in Abaqus
  Extract all displacement snapshots from Abaqus result file ; // in Abaqus
  Collect all snapshots to matrix $\boldsymbol{S}_d$;
  Append $\boldsymbol{S}_d$ to $\boldsymbol{S}$;
$\boldsymbol{A\Sigma B}^T \leftarrow \text{SVD}(\boldsymbol{S})$; // perform SVD on $\boldsymbol{S}$
$\boldsymbol{V} \leftarrow \boldsymbol{A}(:, :n)$; // assign first n columns of matrix $\boldsymbol{A}$ to $\boldsymbol{V}$
return $\boldsymbol{V}$

3.2.2 Co-simulation of the reduced model

A rough overview of the co-simulation with a reduced model is done in the figure 3.2, a theory is explained in 3.1.3. Additional details that apply to our implementation are described here, and a detailed pseudo-code is shown. Each simulation is solved in 30 increments of equal length. A tolerance for a Newton-Raphson procedure (denoted tol in the pseudo-code) is set to $1e-4$ and the maximum number of iterations per increment ($maxiter$) is set to 15. The loads are defined according to 2.2.6. Again, the main workflow runs in Python, while the steps executed in Abaqus are indicated with a comment "// in Abaqus". Some more detailed information about the Abaqus implementation can be found in the appendix A.

Algorithm 2: Co-simulation of the reduced model

Input: Ab_model ... Abaqus model with defined mesh and material properties

LC ... Load case is a dictionary with defined:

$-$ f ... frequency of the load

$-$ X ... amplitude of the load

$-$ ψ ... set of nodes on which the load is applied

$\boldsymbol{V}$ ... reduction basis

n_{inc} ... number of increments

$maxiter$ ... maximal number of iterations in Newton's procedure

tol ... tolerance of the residual forces in Newton's procedure

Output: $\boldsymbol{x}$ ... array of displacements for each dof in each time increment

$\boldsymbol{t}$ ... list of times corresponding to solution increments

initialize $\boldsymbol{x}$; // Empty array for collecting displacement vectors.

initialize $\boldsymbol{q}$; // definition 3.14

initialize $\boldsymbol{t}$; // Empty vector for collecting times of increments.

$\boldsymbol{x}_{n=0} \leftarrow \boldsymbol{0}$; // $\boldsymbol{x}$ at increment 0

$\boldsymbol{q}_{n=0} \leftarrow \boldsymbol{0}$; // $\boldsymbol{q}$ at increment 0, assumed to be 0

$t_{n=0} \leftarrow 0$; // $\boldsymbol{t}$ at increment 0

$t_{end} \leftarrow \frac{1}{f}$; // Analysis' time is set to 1 cycle by default.

$\Delta t_{inc} \leftarrow \frac{t_{end}}{n_{inc}}$; // time step

$\boldsymbol{f}_{stiff,n=0} \leftarrow \boldsymbol{0}$; // definition 3.22

$\boldsymbol{v}_{n=0} \leftarrow \boldsymbol{0}$; // definition 3.22

$\boldsymbol{K}_{n=0} \leftarrow \boldsymbol{K}(\boldsymbol{x}_0)$; // in Abaqus

$\boldsymbol{M}_{n=0} \leftarrow \boldsymbol{M}(\boldsymbol{x}_0)$; // in Abaqus

for $n \leftarrow 0$ **to** n_{inc} **do** // incrementation loop - Backward Euler

  $i \leftarrow 0$;

  $\cdot^{i}_{n+1} \leftarrow \cdot^{i,final}_{n}$; // final values of n become intial guess of $n+1$

  $t_{n+1} \leftarrow t_n + \Delta t_{inc}$;

  $\boldsymbol{f}^{i}_{inert} = \boldsymbol{0}$; // inertia forces reset

  $$\boldsymbol{r}^{i}_{n+1} \leftarrow \begin{bmatrix} \boldsymbol{V}^T \boldsymbol{F}_{ext}(t_{n+1}) \\ \boldsymbol{0} \end{bmatrix} - \begin{bmatrix} \boldsymbol{V}^T \boldsymbol{f}^{i}_{stiff,n+1} \\ \boldsymbol{v}^{i}_{n+1} \end{bmatrix} - \boldsymbol{f}^{i}_{inert,n+1} \; ;$$

  while $i < maxiter \; and \; norm(\boldsymbol{r}^{i}_{n+1}) > tol$ **do** // iteration loop - Newton

    $$\boldsymbol{X}^{i}_{n+1} = \left(\frac{1}{\Delta t_{inc}} \begin{bmatrix} \boldsymbol{V}^T \boldsymbol{M}_{n=0} \boldsymbol{V} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} + \begin{bmatrix} \boldsymbol{0} & \boldsymbol{V}^T \boldsymbol{K}^{i}_{n+1} \boldsymbol{V} \\ -\boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \right) ; \quad \text{// build } \boldsymbol{X}$$

    $\Delta \boldsymbol{q}^{i+1}_{n+1} \leftarrow \boldsymbol{X}^{-1} \boldsymbol{r}^{i}_{n+1}$;

    $\boldsymbol{q}^{i+1}_{n+1} \leftarrow \boldsymbol{q}^{i}_{n+1} + \Delta \boldsymbol{q}^{i+1}_{n+1}$;

    $\boldsymbol{x}^{i+1}_{n+1} \leftarrow \begin{bmatrix} \boldsymbol{0} & \boldsymbol{V} \end{bmatrix} \boldsymbol{q}^{i+1}_{n+1}$; // The second halve of $\boldsymbol{q}^{i+1}_{n+1}$ are $\boldsymbol{x}_{red}$

    $\boldsymbol{K}^{i+1}_{n+1} \leftarrow \boldsymbol{K}(\boldsymbol{x}^{i+1}_{n+1}, t_{n+1})$; // in Abaqus

    $\boldsymbol{f}^{i+1}_{stiff,n+1} \leftarrow \boldsymbol{f}_{stiff}(\boldsymbol{x}^{i+1}_{n+1}, t_{n+1})$; // in Abaqus

    $\boldsymbol{v}^{i+1}_{n+1} \leftarrow \boldsymbol{v}^{i}_{n+1} + \begin{bmatrix} -\boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \Delta \boldsymbol{q}^{i+1}_{n+1}$; // update $\boldsymbol{v}$

    $$\boldsymbol{f}^{i+1}_{inert} \leftarrow \boldsymbol{f}^{i}_{inert} + \frac{1}{\Delta t_{inc}} \begin{bmatrix} \boldsymbol{V}^T \boldsymbol{M} \boldsymbol{V} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \Delta \boldsymbol{q}^{i+1}_{n+1} \; ; \quad \text{// update } \boldsymbol{f}_{inert}$$

    $$\boldsymbol{r}^{i+1}_{n+1} \leftarrow \begin{bmatrix} \boldsymbol{V}^T \boldsymbol{F}_{ext}(t_{n+1}) \\ \boldsymbol{0} \end{bmatrix} - \begin{bmatrix} \boldsymbol{V}^T \boldsymbol{f}^{i+1}_{stiff,n+1} \\ \boldsymbol{v}^{i+1}_{n+1} \end{bmatrix} - \boldsymbol{f}^{i+1}_{inert,n+1} \; ;$$

    $i \leftarrow i+1$

  $\boldsymbol{x}(n+1) \leftarrow \boldsymbol{x}^{i}_{n+1}$; // save last solution in $\boldsymbol{x}$ array

  $t(n+1) \leftarrow t_{n+1}$; // save time in $\boldsymbol{t}$ array

return $\boldsymbol{x}$, $\boldsymbol{t}$

3.3 Training, Result, Discussion

In order to systematically show the behavior of the POD reduced model, several investigations will be shown in this section. For better understanding, the definitions of the training loads, validation loads and the results and comments will be presented in parallel. Two sets of load parameters are used for each investigation:

- Training cases - loads used to calculate the reduction basis.
- Validation cases - loads used to validate the reduced model in a co-simulation

approach.

Wherever a distinction is made between Case A and B, it means that Case A represents behavior with a poor set of parameters or training cases. Case B is an improvement which adequately captures the behavior.

Remark
In some comparisons, the same number n (the dimension of the reduced model) is used, regardless of the different dimension of the training data. This may be considered an unfair comparison, as it may not be an optimal value for all models being compared. It is good to keep this in mind when looking at the results. However, with a constant value n, the models are equally reduced and well comparable in this aspect. For this reason, at many places in this thesis the same number n is assumed for the models being compared.

3.3.1 Load of different frequencies

This subsection discusses how to create a model that captures the behavior at different frequencies. A distinction is made between Case A and Case B. Case A is unsuccessful, Case B is successful. They are using different set of training cases.

Training loads

As explained in the load case definition in 2.2.6, each load case has its own amplitude X, frequency f and set of excitation nodes ψ. The properties of the training loads are presented in tables 3.1 and 3.2, which shows training simulations for Case A and Case B, respectively.

Tr. case	$f[Hz]$	$X[N]$	ψ
1	101	0.5	[205]

Table (3.1) Training loads, different frequencies, Case A

Tr. case	$f[Hz]$	$X[N]$	ψ
1	1	0.5	[205]
2	21	0.5	[205]
3	41	0.5	[205]
4	61	0.5	[205]
5	81	0.5	[205]
6	101	0.5	[205]

Table (3.2) Training loads, different frequencies, Case B

Reduced model

For each Case, A and B, a reduced model was created. It was done with 12 singular vectors of the matrix $\boldsymbol{A}$ from SVD ($n = 12$).

Validation loads

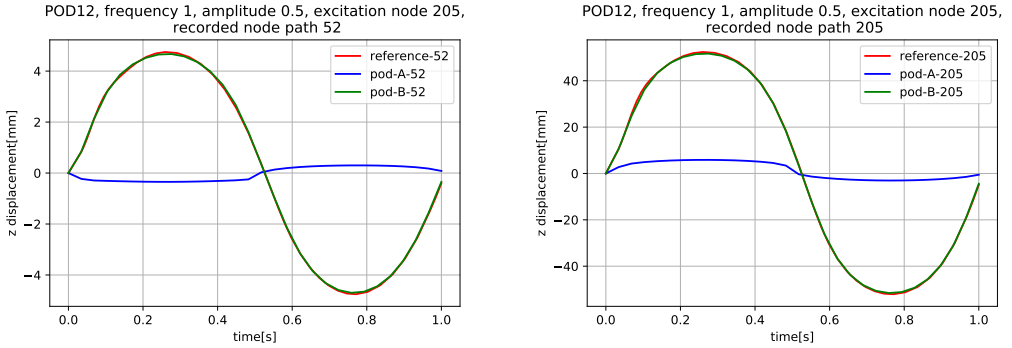
Both models, A and B, use the same validation cases. The validation cases are listed in the table below.

Vl. case	$f[Hz]$	$X[N]$	ψ
1	1	0.5	[205]
2	31	0.5	[205]
3	51	0.5	[205]
4	91	0.5	[205]
5	101	0.5	[205]

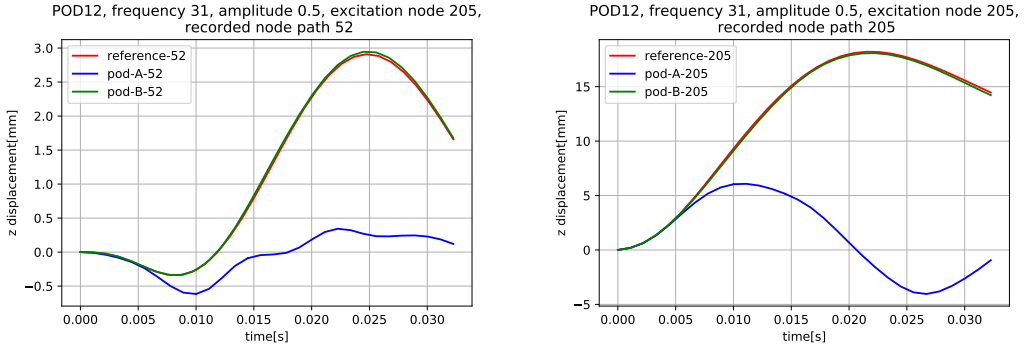
Table (3.3) Validation loads, different frequencies, Case A and B

Results

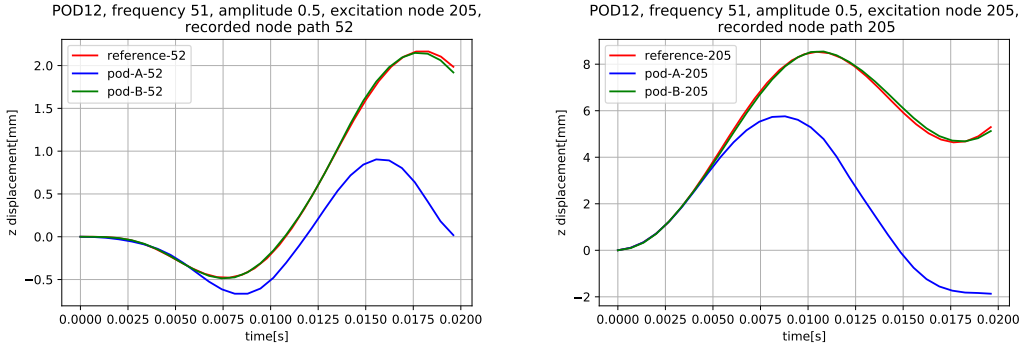
Here, the paths for two nodes (52 and 205) were recorded, to get better spatial understanding of what is happening with a model. The "z" coordinate is plotted in the diagrams. For each load case, 2 plots are shown - the path for node 52 and 205. Different colors represent different solutions: reference, Case A and Case B.



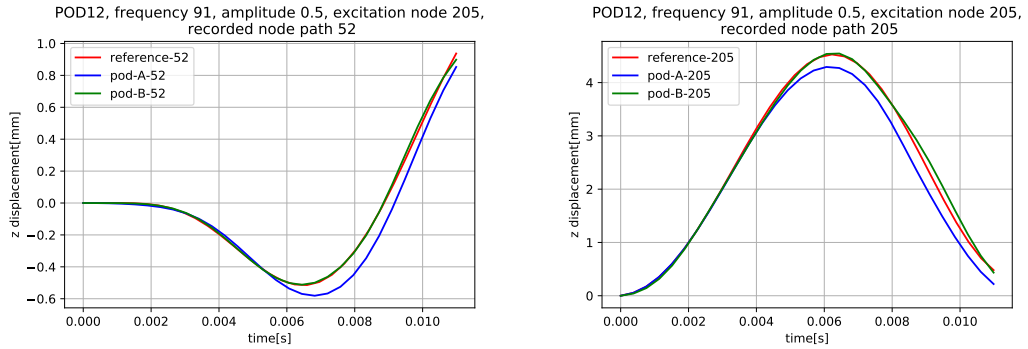
(a) Vl. case 1



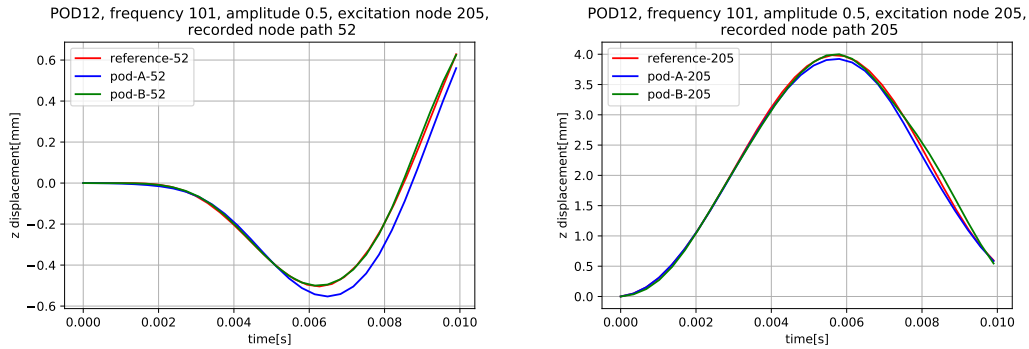
(b) Vl. case 2



(c) Vl. case 3



(d) VI. case 4



(e) VI. case 5

Figure (3.3) Different frequencies

Discussion

The starting idea for Case A was that the highest frequency of interest maybe cover behavior of the complete interval from zero to that frequency. The basis for such an idea was that at higher frequencies, the lower modes are generally also present. With training of the model at the highest frequency, entire spectrum of the modes should be covered then. However, as can be seen from the results, the quality of the model depends on how close the frequencies for training and validation are. For Case A, it can be seen that the discrepancy between the reference and POD solutions increases as the difference between the training and validation frequencies increases. Since the training simulation was run for 101 Hz, the figure 3.3e seems to model behavior almost perfectly. Going from that frequency away, every figure gets worse: 3.3d, 3.3c, 3.3b, 3.3a.

According to the results seen in Case A, Case B was performed with training simulations at several frequencies. In the figures above, the improvement can be clearly seen. In our case, the step of 20 Hz within a frequency range of interest (1 ... 101 Hz) ensures a good solution.

Here two possible explanations are given why our hypothesis, that only the highest frequency could describe the whole behavior in a frequency range is not correct.

1. SVD tool is a statistical tool used in POD calculation. For the reduction basis V, only some of the most important modes for SVD are used. It could be that the lowest modes are present in the high frequency training simulation, but they are not involved enough to be identified (statistically) as important modes for the solution. Thus, they are present in the left singular matrix of the SVD, but not in the most important locations to be included later in the reduction matrix. The solution is, to run additional training simulations in which other modes are of grater importance. In

doing so, the overall important vectors for all the frequencies will be chosen to be included in a reduction basis later.

2. Another possible reason is that it is not possible to capture the general time-dependent behavior at only one frequency. The properties of the model (the stiffness) is time-dependent. Consequently, the modal behavior is also frequency-dependent.

3.3.2 Load of different amplitudes

The dynamics of our system is amplitude dependent, since nonlinear elastic material (hyperelasticity) and geometrical nonlinearity are considered. In this subsection, we investigate how to build a reduced model capable of capturing the behavior of different amplitudes. Again, a distinction is made between Case A and B. Case A (unsuccessful) and B (successful) will be presented in parallel.

Training loads

As explained in the load case definition in 2.2.6, each load case has its own amplitude X, frequency f and set of excitation nodes ψ. The properties of the training loads are presented in tables 3.4 and 3.5, which show training simulations for Case A and Case B, respectively.

Tr. case	$f[Hz]$	$X[N]$	ψ
1	101	1.5	[205]

Table (3.4) Training loads, different amplitudes, Case A

Tr. case	$f[Hz]$	$X[N]$	ψ
1	101	0.1	[205]
2	101	0.5	[205]
3	101	1.0	[205]
4	101	1.5	[205]

Table (3.5) Training loads, different amplitudes, Case B

Reduced model

Two reduced models were created again. One for Case A and one for Case B. Both use 12 singular vectors for the basis $\boldsymbol{V}$. ($n = 12$)

Validation loads

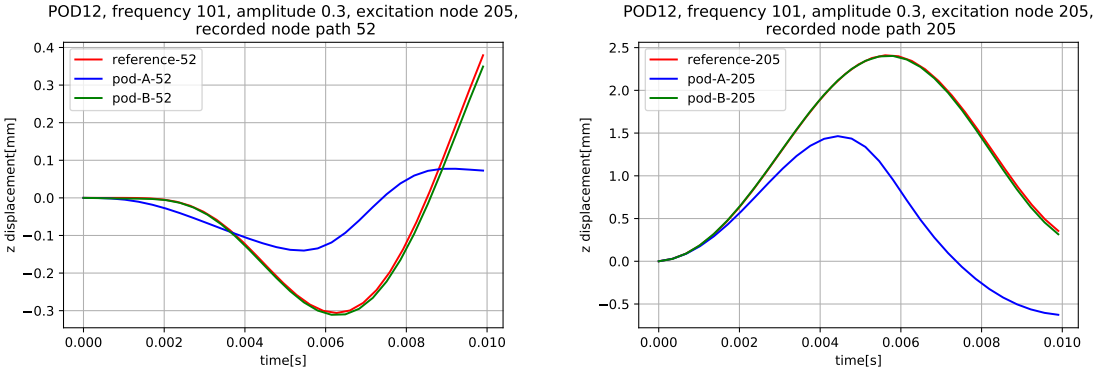
Load cases are defined according to the definition in 2.2.6. Case A and B use the same validation cases to be well comparable. They are defined in a table 3.6.

Vl. case	$f[Hz]$	$X[N]$	ψ
1	101	0.3	[205]
2	101	0.7	[205]
3	101	1	[205]
4	101	1.4	[205]
5	101	1.5	[205]

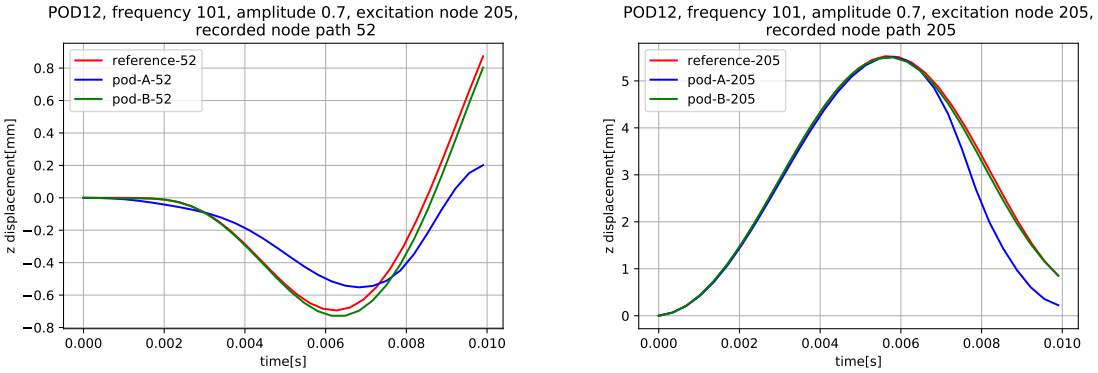
Table (3.6) Validation loads, different amplitudes, Case A and B

Results

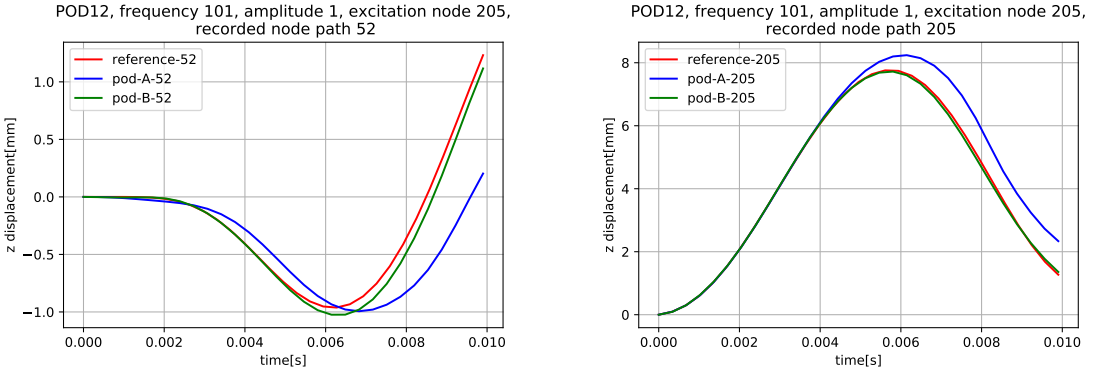
Here, the paths for two nodes (52 and 205) were recorded, to get better spatial understanding of what is happening with a model. The "z" coordinate is plotted in the diagrams. For each load case, 2 plots are shown - the path for node 52 and 205. Different colors represent different solutions: reference, Case A and Case B.



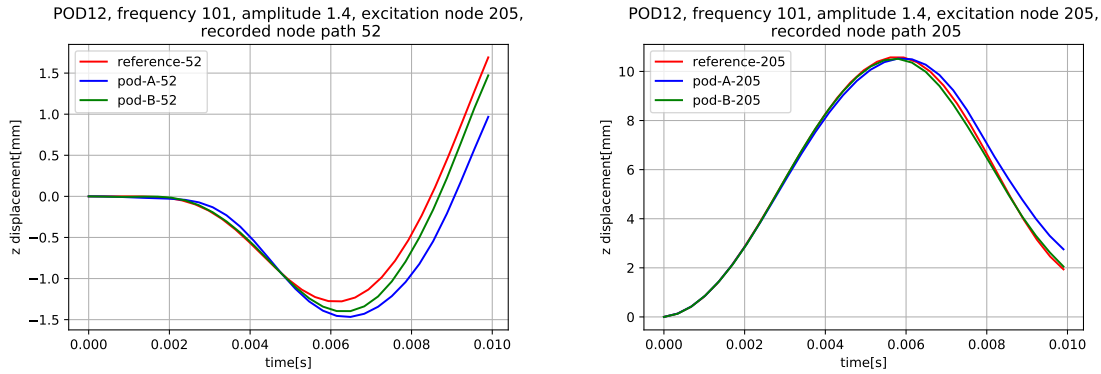
(a) Vl. case 1



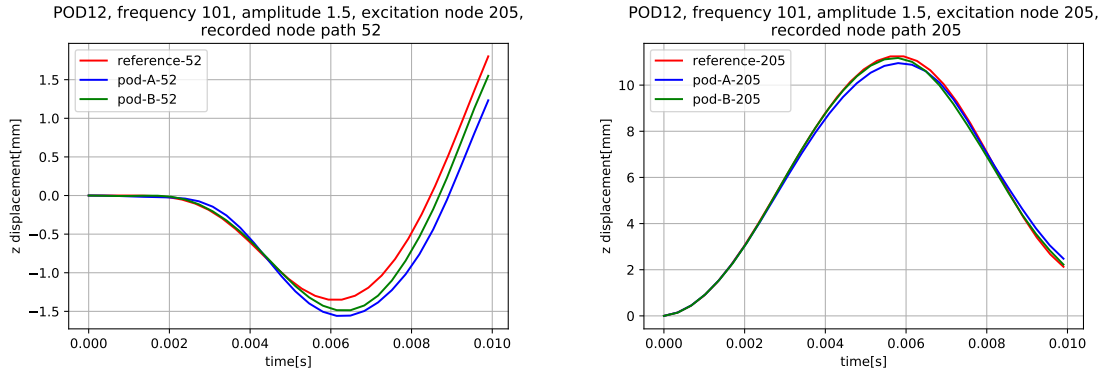
(b) Vl. case 2



(c) Vl. case 3



(d) VI. case 4



(e) VI. case 5

Figure (3.4) Different amplitudes

Discussion

The starting idea was the same as for the frequency dependence. The idea was that the highest excitation amplitude should generally include almost all possible deformations of the lower amplitudes. However, a look at the results clearly shows that the model gives useful results for the amplitudes that are close to the training amplitude. Further we move away from training simulations, worse approximation the reduced model brings.

Case B provides the solution to the problem. It is a more "dense" distribution of the training simulations in terms of amplitudes. Compared to the results of Case A, a significant improvement can be observed. In Case B all the validation cases are captured quality. A possible intuitive explanation is similar to that one, already discussed by different frequencies. It is possible that many possible deformation shapes are present in a high amplitude simulation. However, they are not sorted according to the overall importance for any amplitude, but only according to the importance for this training simulation. For another training simulation, the importance could be different. The solution is to add additional training simulations with different amplitudes to balance the importance of the different modes under different loads.

It is good to know, that the need of running several training simulations with different amplitudes is coming from the nonlinearities of the model (material and geometrical). In case of purely linear system, also the Case A may be ok, since deformation shapes in singular vectors are usually combinations of physical modes.

3.3.3 Validation load on another node than training load

So far, the validation load has been set to the same node as the training load. In this subsection, the generality of the fitted model is proven. The validation load is set to a different node than the training load.

Three cases are considered here. Case A is the unsuccessful training of the model. Case B is the successful training of the model. In Case C, an attempt was made to simplify and speed up the computation, but it has barely achieved an improvement in comparison to the Case A.

Training loads

As explained in the load case definition in 2.2.6, each load case has its own amplitude X, frequency f and set of excitation nodes ψ. The properties of the training loads are presented in tables 3.7, 3.8 and 3.9 for Case A, B and C, respectively. In Case A, one training simulation is performed with load on one node. In Case B, multiple training simulations are performed on different nodes. In Case C, one training is performed on multiple nodes.

Tr. case	$f[Hz]$	$X[N]$	ψ
1	101	0.5	[205]

Table (3.7) Training loads, Validation node differs from training node, Case A

Tr. case	$f[Hz]$	$X[N]$	ψ
1	101	0.5	[21]
2	101	0.5	[64]
3	101	0.5	[92]
4	101	0.5	[125]
5	101	0.5	[142]
6	101	0.5	[175]
7	101	0.5	[181]
8	101	0.5	[205]

Table (3.8) Training loads, Validation node differs from training node, Case B

Tr. case	$f[Hz]$	$X[N]$	ψ
1	101	0.5	[21, 64, 92, 125, 142, 175, 181, 205]

Table (3.9) Training loads, Validation node differs from training node, Case C

Reduced model

For each of Cases A, B, and C, a model was built based on training simulations used in each case. All three use 12 singular vectors in a reduction basis $\boldsymbol{V}$. ($n = 12$)

Validation loads

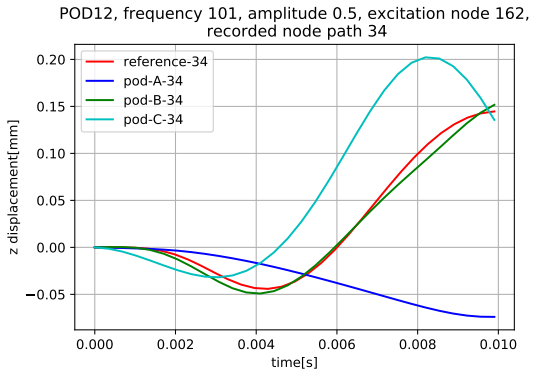
The validation case used here has the same amplitude and frequency as all trainings, only the node of excitation is different.

Vl. case	$f[Hz]$	$X[N]$	ψ
1	101	0.5	[162]

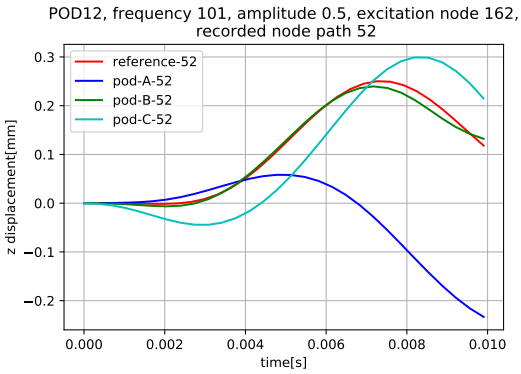
Table (3.10) Validation loads, Validation node differs from training node, Case A, B and C

Results

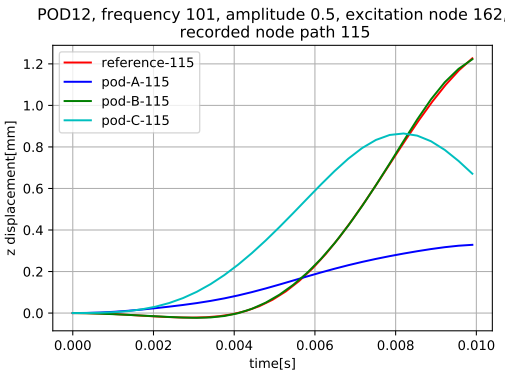
Here, paths for six nodes (34, 52, 115, 152, 183 and 205) were recorded, to provide a better spatial understanding of what is happening in a model. For each recorded node, a plot was generated. On each plot, there are four solutions in four colors; reference, Case A, Case B and Case C.



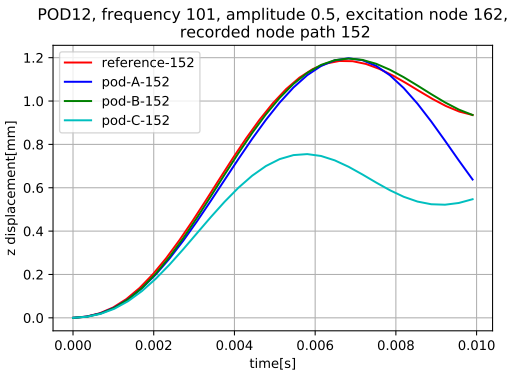
(a) Vl. case 1, node 34



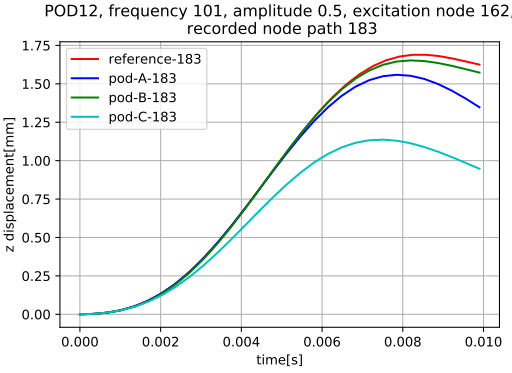
(b) Vl. case 1, node 52



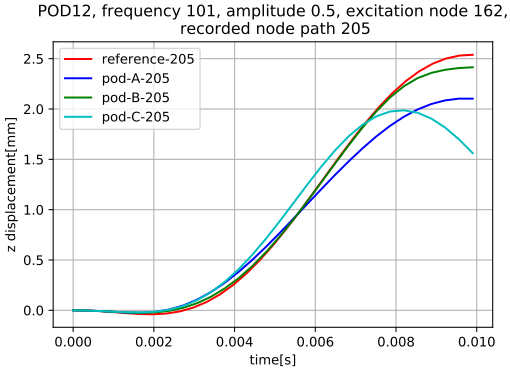
(c) Vl. case 1, node 115



(d) Vl. case 1, node 152



(e) Vl. case 1, node 183



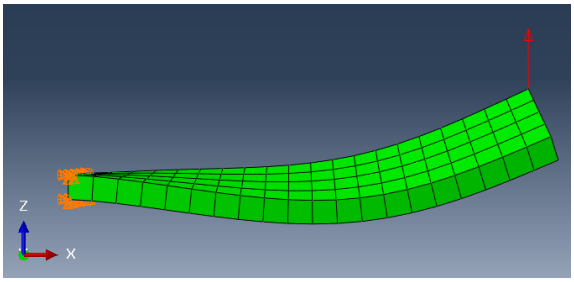
(f) Vl. case 1, node 205

Figure (3.5) Validation node differs from training node

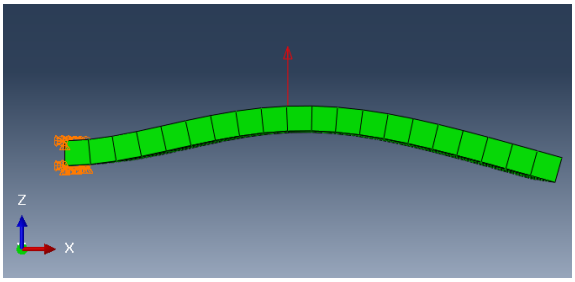
Discussion

It is clearly seen that the trained model in Case A does not capture behavior well. This confirms a statement that the training simulations have to be something similar what the model should be capable of later. In Case B, a significant improvement is achieved. In Case C all the loads applied in Case B were superimposed in one training simulation to reduce training costs. Case C does not really improve the behavior of the model compared to A. A possible explanation is given in the next paragraph.

Look at the two exemplary deformation states caused by a force applied at two different locations. The forces are applied in two separate simulations, as in Case B. For the first deformation state, the load is applied to node 205, for the second to node 93.



(a) Exemplary deformation state 1 - load on node 205



(b) Exemplary deformation state 2 - load on node 93

When both forces from the above cases are superimposed in one simulation, the deformation state is shown in the figure below. This could be understood as a snapshot obtained in Case C. Intuitively, deformation state 3 can be composed of states 1 and 2.

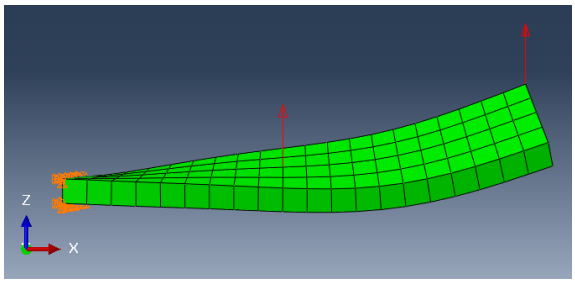


Figure (3.7) Exemplary deformation state 3 - load on node 93 and 205

For ease of further explanation, it is assumed that all three deformation states showed above are directly included in the reduction basis. The first two are in the basis of Case B, the third one is in the reduction basis of Case C. Case B can consequently capture a behavior with a force at node 93, at node 205 or at both nodes at the same time. On the other hand, in the basis of Case C, there is just a third deformation state present. With only this deformation state, we cannot describe the behavior of the mode if the load is applied either at node 93 or node 205, but only the behavior at both nodes at the same time. This could be a reason why Case C is not successful.

3.3.4 Different number of reduction vectors n

So far, we have assumed that the number of singular vectors in the reduced basis is $n = 12$. This was sufficient to capture what we wanted to show. Here an influence of the number of vectors on the quality of the solution is investigated.

Training loads

As explained in the load case definition in 2.2.6, each load case has its own amplitude X, frequency f and set of excitation nodes ψ. The properties of the training loads are presented in tables 3.11.

Tr. case	$f[Hz]$	$X[N]$	ψ
1	101	0.5	[21]
2	101	0.5	[64]
3	101	0.5	[92]
4	101	0.5	[125]
5	101	0.5	[142]
6	101	0.5	[175]
7	101	0.5	[181]
8	101	0.5	[205]

Table (3.11) Training loads, Different number of reduction vectors

Reduced model

Five different reduced models were created here. All are based on the same training simulations, but use different n - number of the basis vectors present in $\boldsymbol{V}$ or in other words the dimension of the reduced model. The numbers for n are following: 2, 4, 8, 12 and 16.

Validation loads

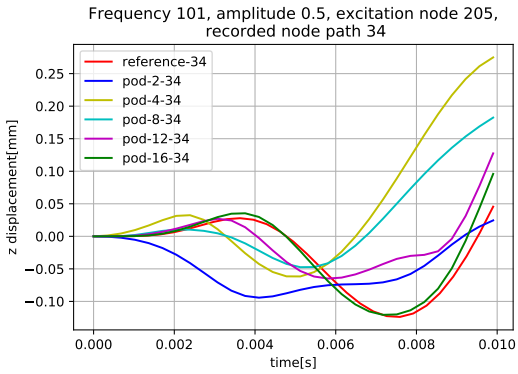
Here is only one validation case, which will be applied to all models. The parameters for this case are defined in the table 3.12.

Vl. case	$f[Hz]$	$X[N]$	ψ
1	101	0.5	[203]

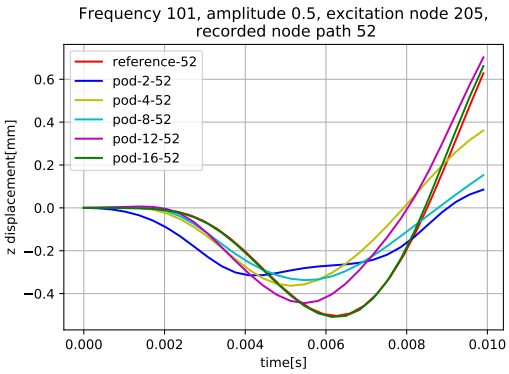
Table (3.12) Validation load, Different number of reduction vectors

Results

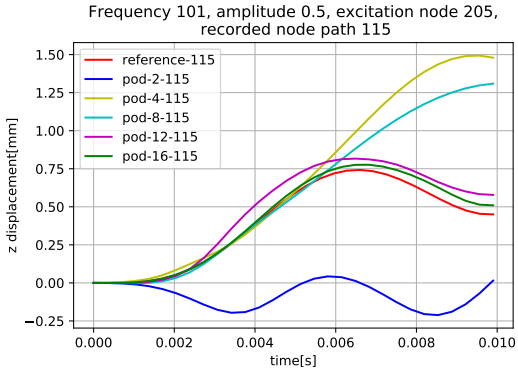
Here, paths for six nodes (34, 52, 115, 152, 183 and 205) were recorded, to provide a better spatial understanding of what is happening with a model. There are six plots, for each recorded node one. On each graph, there are six solutions in different colors; reference, pod-2, pod-4, pod-8, pod-12, pod-16. The numbers denote a reduced dimension of the model (n).



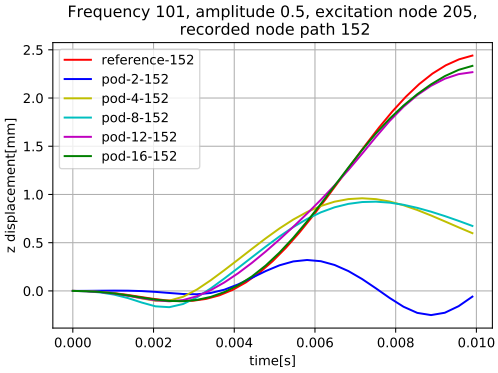
(a) Behavior on node 34



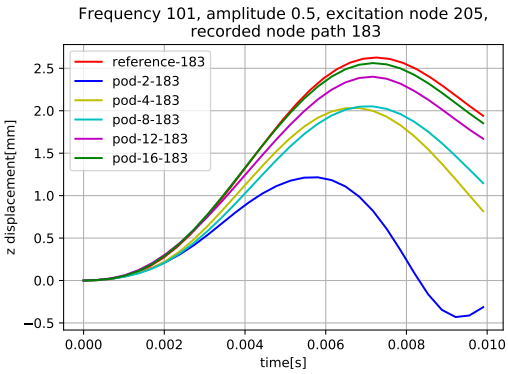
(b) Behavior on node 52



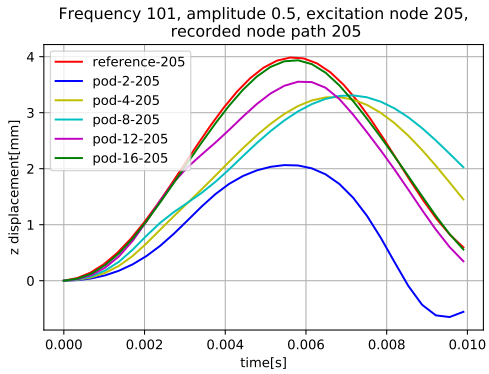
(c) Behavior on node 115



(d) Behavior on node 152



(e) Behavior on node 183



(f) Behavior on node 205

Figure (3.8) Different number of reduction vectors

Discussion

From the results above, it can be seen that more vectors mean a better approximation. This can be seen especially well in the figure 3.8e. This is somehow expected. With more degrees of freedom an approximation becomes better.

3.3.5 General example

Here an example of a general trained model is given. There are 24 training cases that attempt to capture frequency, amplitude and spatial distribution of loads. Afterwards, 10

arbitrarily selected validation cases are run in a co-simulation, to prove the quality of the reduced model.

Training loads

The properties for all 24 training loads are presented in the table 3.13.

Tr. case	$f[Hz]$	$X[N]$	ψ
1	1	0.1	[21]
2	1	0.1	[205]
3	61	0.1	[64]
4	61	0.1	[142]
5	81	0.1	[92]
6	81	0.1	[205]
7	101	0.1	[21]
8	101	0.1	[181]
9	1	0.8	[64]
10	1	0.8	[125]
11	21	0.8	[92]
12	21	0.8	[175]
13	41	0.8	[64]
14	41	0.8	[205]
15	81	0.8	[125]
16	81	0.8	[175]
17	21	1.5	[21]
18	21	1.5	[92]
19	41	1.5	[64]
20	41	1.5	[175]
21	61	1.5	[142]
22	61	1.5	[205]
23	101	1.5	[181]
24	101	1.5	[205]

Table (3.13) General training loads

Reduced model

Only one reduced model is used here. It is based on training simulations listed in the table 3.13. The dimension of the reduced model is set to $n = 16$.

Validation loads

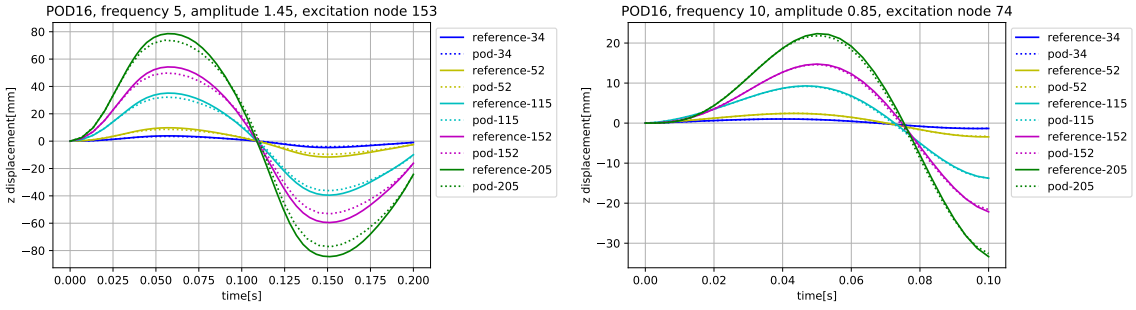
Validation loads are listed in the table below.

Vl. case	$f[Hz]$	$X[N]$	ψ
1	5	1.45	[153]
2	10	0.85	[74]
3	25	1.1	[34]
4	35	0.95	[182]
5	40	0.6	[51]
6	55	1.3	[203]
7	60	0.3	[114]
8	75	0.45	[135]
9	80	1.25	[165]
10	95	0.1	[92]

Table (3.14) General validation loads

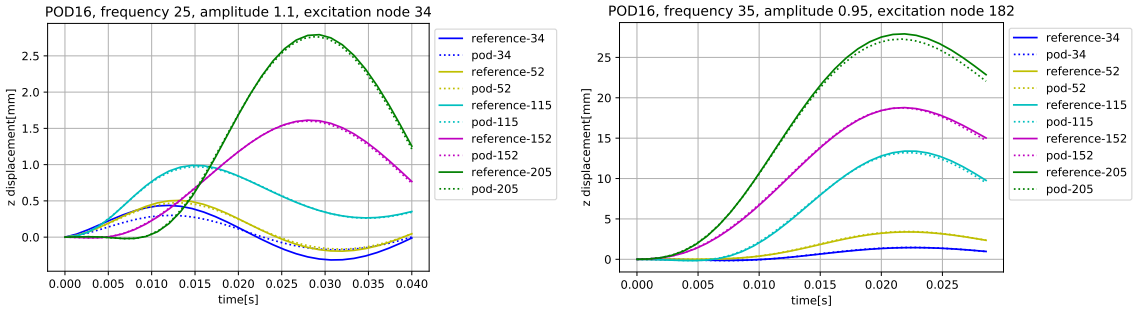
Results

Here, paths for five nodes (34, 52, 115, 152 and 205) were recorded to provide better spatial understanding of what is happening with a model. There are 10 plots, one for each validation case. In each graph, the paths of five nodes are shown in different colors.



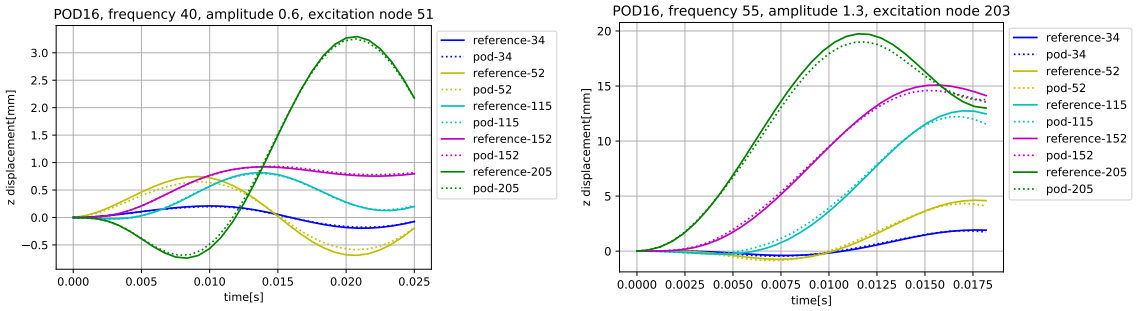
(a) Validation case 1

(b) Validation case 2



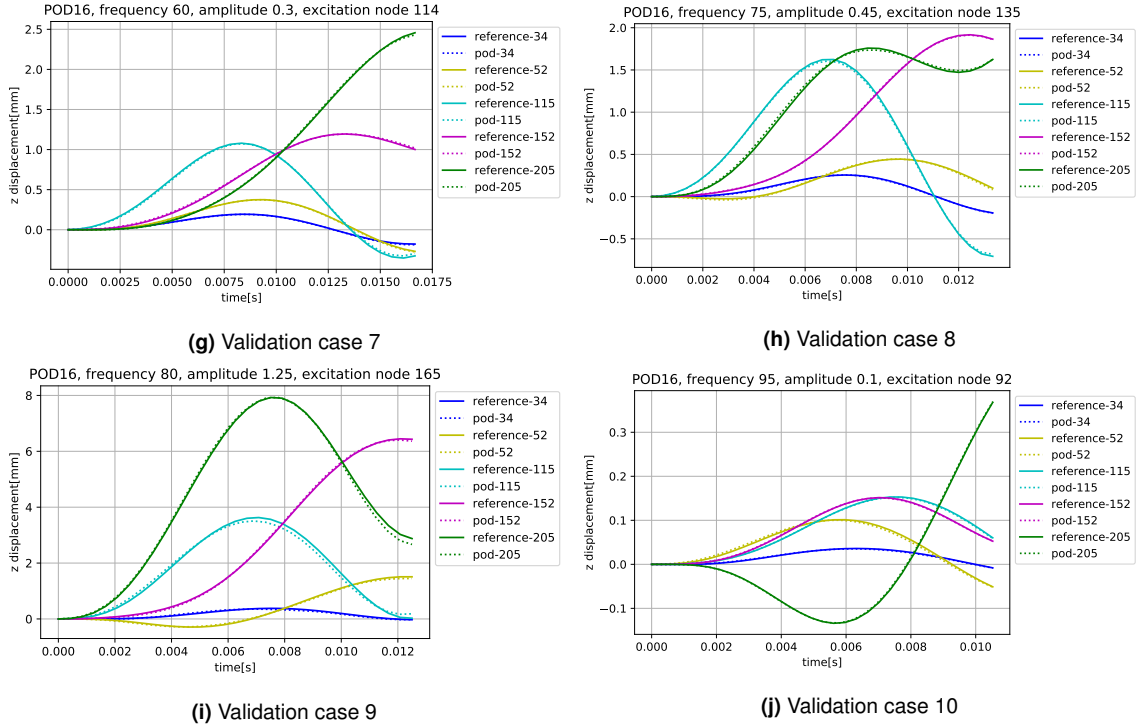
(c) Validation case 3

(d) Validation case 4



(e) Validation case 5

(f) Validation case 6



(g) Validation case 7

(h) Validation case 8

(i) Validation case 9

(j) Validation case 10

Figure (3.9) General validation cases - results

Discussion

From the results above it can be seen that the reduction works as expected. The results of the reduced model are within reasonable tolerance. The degrees of freedom of the model were reduced from 630 to 16. A reduction is judged as successful. Maybe the reduction is not so impressive. But we have to be aware that the number of the vectors needed could be driven by the geometry and type of the loads (physical predispositions) and is independent from the mesh. Even if much denser mesh would be used, the number of POD singular vectors required would remain in the same order of magnitude. In that case the reduction would look much more efficient.

3.4 Outlook

If the linear system would be in our scope, the projection based reduction would work efficiently. The matrices $\boldsymbol{M}_{red} = \boldsymbol{V}^T \boldsymbol{M} \boldsymbol{V}$ and $\boldsymbol{K}_{red} = \boldsymbol{V}^T \boldsymbol{K} \boldsymbol{V}$ would be evaluated once. After that, the calculation would be performed on the system of reduced dimensions. In our case, however, a nonlinear system is in the scope. Consequently, this means that the matrix $\boldsymbol{K}(\boldsymbol{x}, t)$ is constantly changing and the forces $\boldsymbol{f}_{stiff}(\boldsymbol{x}, t)$ must be updated in each iteration. Since the nonlinear dependence of the stiffness matrix and the forces is directly known only in the spatial coordinates, a reduced displacement vector $\boldsymbol{x}_{red}$ must be transformed into the original one ($\boldsymbol{x}$) in each iteration, and then the displacement is applied to the full system. Then a new stiffness matrix and forces are extracted. Afterwards the matrix and force vector are projected back into reduce space. This is, of course, an expensive procedure where saving computational time is not granted. How the problem tries to be alleviated will be discussed in the next chapter (4).

Chapter 4

Hyper-Reduction

In this chapter, the content of hyper-reduction and ECSW in particular is presented. First, some theoretical background is introduced, then the Python-Abaqus co-simulation implementation is explained, and finally the results are shown along with discussion. At the very end of the chapter, there is a section explaining an alternative formulation of ECSW - the Work-based ECSW calculation.

4.1 Theoretical background

4.1.1 General

As shown in the results and as described in the outlook of the previous chapter (3), POD works well but has an important drawback when applied to a nonlinear system. The main drawback is the remaining expensive computation of the tangential stiffness matrix and the nonlinear forces. Hyper-reduction methods try to solve this problem.

According to the overview in [17], there are several hyper-reduction methods such as Polynomial tensors (e.g. [14]), Discrete empirical interpolation method [6] or Energy conserving sampling and weighting (ECSW) method [3]. Each of these methods attempts to achieve a reduction in a different way. Polynomial tensors tries to capture the nonlinear behavior with an expansion in a Taylor series. Discrete empirical interpolation attempts to describe the nonlinearity of the forces with a reduced basis. The concept is similar as projection discussed before on the displacement field, now just applied to the nonlinear forces. The ECSW tries to reduce the space-domains calculated only to the most representative ones (select some representative elements and evaluate the stiffness and the nonlinear forces there). Then, the result for the full domain is recovered by appropriately weighting the calculated domains.

According to the comparison done in [17], ECSW seems to be the most promising method since it offers some advantages such as symmetry and stability preservation. For this reason, this method is also used in this thesis.

4.1.2 Energy Conserving Sampling and Weighting (ECSW)

The main idea of ECSW is to reduce the cost of the calculation of nonlinear stiffness matrix and nonlinear forces in each iteration. Let us call the entire set of analyzed elements ϵ.

Instead of applying a new displacement field to all elements of the high fidelity model ϵ in each iteration and extracting the entries for the stiffness and nonlinear forces, only some of the most representative elements of the high-fidelity model are selected (let us call them a subset ϵ_r) and the stiffness and nonlinear forces are evaluated there. The stiffness properties and nonlinear forces are then extended to the entire domain ϵ, with an appropriate weighting the values evaluated on ϵ_r. The unselected elements are thus assumed to have the properties expressed in a linear combination of the selected elements - those from the subset ϵ_r. Referring to [17] the ecsw reduced stiffness forces and the stiffness matrix are calculated using the equations 4.1 and 4.2 respectively.

$$\boldsymbol{f}_{stiff,ECSW}(\boldsymbol{x}_{red}) = \sum_{e \in \epsilon_r} \xi_e \boldsymbol{V}^T \boldsymbol{L}_e^T \boldsymbol{f}_e(\boldsymbol{L}_e \boldsymbol{V} \boldsymbol{x}_{red}) \tag{4.1}$$

$$\boldsymbol{K}_{ECSW}(\boldsymbol{x}_{red}) = \sum_{e \in \epsilon_r} \xi_e \boldsymbol{V}^T \boldsymbol{L}_e^T \boldsymbol{K}_e(\boldsymbol{L}_e \boldsymbol{V} \boldsymbol{x}_{red}) \boldsymbol{L}_e \boldsymbol{V} \tag{4.2}$$

In the equations above, $\boldsymbol{V}$ is a reduction matrix, $\boldsymbol{f}_e$ is an elemental stiffness force, $\boldsymbol{K}_e$ is an elemental stiffness matrix, $\boldsymbol{x}_{red}$ is the vector of the reduced displacement (with relation to the full displacement field $\boldsymbol{x} = \boldsymbol{V}\boldsymbol{x}_{red}$), ξ_e is a scalar weight for an element e and $\boldsymbol{L}_e$ is a localization matrix providing the mapping of the local (elemental) vector to the global one.

An important part of the method is selection criterion (which elements from ϵ go into the subspace ϵ_r) and the calculation procedure for the element weights $\boldsymbol{\xi}$ (where $\boldsymbol{\xi}$ is a vector of weights for all elements). The first condition in ECSW, is an equality of the virtual works evaluated for the original set of elements ϵ (all elements of the high fidelity model) and the subset of "representative" elements ϵ_r.

$$\delta \boldsymbol{W}_r = \sum_{e \in \epsilon} \delta \boldsymbol{x}_{red}^T \boldsymbol{V}^T \boldsymbol{L}_e^T \boldsymbol{f}_e(\boldsymbol{L}_e \boldsymbol{V} \boldsymbol{x}_{red}) = \sum_{e \in \epsilon_r} \xi_e \delta \boldsymbol{x}_{red}^T \boldsymbol{V}^T \boldsymbol{L}_e^T \boldsymbol{f}_e(\boldsymbol{L}_e \boldsymbol{V} \boldsymbol{x}_{red}) \tag{4.3}$$

Equation 4.3 has one trivial solution which is not of our interest. It is $\boldsymbol{\xi}$ being vector of ones. This would mean that no reduction is obtained. For a reduction in general the second equality must be replaced by an approximate equality up to a certain user-defined tolerance τ. In addition, the premultiplication with a virtual displacement $\delta \boldsymbol{x}_{red}^T$ could be omitted since it is present on both sides of the equation.

$$\sum_{e \in \epsilon} \boldsymbol{V}^T \boldsymbol{L}_e^T \boldsymbol{f}_e(\boldsymbol{L}_e \boldsymbol{V} \boldsymbol{x}_{red}) \approx \sum_{e \in \epsilon_r} \xi_e \boldsymbol{V}^T \boldsymbol{L}_e^T \boldsymbol{f}_e(\boldsymbol{L}_e \boldsymbol{V} \boldsymbol{x}_{red}) \tag{4.4}$$

To achieve the reduction, an error is introduced. The error must be less than or equal to the defined tolerance τ. The condition is expressed in equation 4.5, where the term $||\dots||_2$ represents the L^2 norm.

$$||\boldsymbol{Y}\boldsymbol{\xi} - \boldsymbol{b}||_2 \leqslant \tau ||\boldsymbol{b}||_2 \tag{4.5}$$

An additional condition is imposed on the vector of weights. All the weights must be zero or positive. This condition ensures the property of stability, which is important for the calculation. The procedure for the calculation of the weights is called Sparse Non-Negative Least Square (sNNLS) and is well explained in [17] with a pseudocode and graphically. Briefly, it is an iterative procedure that starts with an empty subset of the elements ϵ_r. In each iteration it looks for the next most representative element (the one, that will reduce the error in 4.5 the most if added to ϵ_r) and calculates the optimal positive weight for it. As

more elements are added, the result becomes more accurate - the discrepancy between the left and right sides of the equation 4.4 becomes smaller. As soon as it is smaller than τ, the procedure ends. Consequently, smaller τ means more elements. In the extreme case with $\tau = 0$ all elements are chosen with a weight of one (no reduction obtained).

Following the procedure presented in [17], the only inputs in the standalone algorithm of sNNLS are the matrix $\boldsymbol{Y}$, the vector $\boldsymbol{b}$ and τ.

$$\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{V}^T\boldsymbol{L}_1^T\boldsymbol{f}_1(\boldsymbol{L}_1\boldsymbol{V}\boldsymbol{x}_{red,1}) & \dots & \boldsymbol{V}^T\boldsymbol{L}_\epsilon^T\boldsymbol{f}_\epsilon(\boldsymbol{L}_\epsilon\boldsymbol{V}\boldsymbol{x}_{red,1}) \\ \vdots & \ddots & \vdots \\ \boldsymbol{V}^T\boldsymbol{L}_1^T\boldsymbol{f}_1(\boldsymbol{L}_1\boldsymbol{V}\boldsymbol{x}_{red,ts}) & \dots & \boldsymbol{V}^T\boldsymbol{L}_\epsilon^T\boldsymbol{f}_\epsilon(\boldsymbol{L}_\epsilon\boldsymbol{V}\boldsymbol{x}_{red,m}) \end{bmatrix} \tag{4.6}$$

$$\boldsymbol{b} = \begin{bmatrix} \sum_{e\in\epsilon} \boldsymbol{V}^T\boldsymbol{L}_e^T\boldsymbol{f}_e(\boldsymbol{L}_e\boldsymbol{V}\boldsymbol{x}_{red,1}) \\ \vdots \\ \sum_{e\in\epsilon} \boldsymbol{V}^T\boldsymbol{L}_e^T\boldsymbol{f}_e(\boldsymbol{L}_e\boldsymbol{V}\boldsymbol{x}_{red,ts}) \end{bmatrix} \tag{4.7}$$

The matrix $\boldsymbol{Y}$ has dimension $n \cdot s \cdot ts \times |\epsilon|$, where n is the number of degrees of freedom in the reduced model, s is the number of snapshots per training simulation, ts is the total number of training simulations, and $|\epsilon|$ is the number of elements in the original model. The vector $\boldsymbol{b}$ has length $n \cdot s \cdot ts$. With $\boldsymbol{Y}$ and $\boldsymbol{b}$, the problem can be formulated as:

$$\boldsymbol{Y}\xi \approx \boldsymbol{b} \tag{4.8}$$

Equation 4.8 is thus solved using the sNNLS algorithm which yields ϵ_r and ξ.

As explained at the beginning: once ξ is calculated, the nonlinear stiffness forces and the linearized stiffness matrix could be extracted only on the elements in the subset ϵ_r and later extended to the entire domain ϵ with equations 4.1 and 4.2.

4.1.3 Solving procedure

The solving procedure for the hyper-reduced model is the same as for the reduced one. It is shown in 3.1.3. The fundamental equations 3.30 and 3.33 have to be recalled.

$$\left(\frac{1}{\Delta t_{inc}}\begin{bmatrix}\boldsymbol{V}^T\boldsymbol{M}\boldsymbol{V} & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{I}\end{bmatrix} + \begin{bmatrix}\boldsymbol{0} & \boldsymbol{V}^T\boldsymbol{K}^i_{n+1}\boldsymbol{V}\\ -\boldsymbol{I} & \boldsymbol{0}\end{bmatrix}\right)\Delta\boldsymbol{q}^{i+1}_{n+1} = \begin{bmatrix}\boldsymbol{V}^T\boldsymbol{F}_{ext,n+1}\\ \boldsymbol{0}\end{bmatrix} - \begin{bmatrix}\boldsymbol{V}^T\boldsymbol{f}^i_{stiff,n+1}\\ \boldsymbol{v}^i_{n+1}\end{bmatrix} - \begin{bmatrix}\boldsymbol{f}^i_{inert}\end{bmatrix} \tag{3.30 revisited}$$

$$\boldsymbol{X}^i_{n+1}\Delta\boldsymbol{q}^{i+1}_{n+1} = \boldsymbol{r}^i_{n+1} \tag{3.33 revisited}$$

The only difference is in evaluation of the stiffness matrix and nonlinear forces. They are evaluated according to the theory presented above. The equations 4.1 and 4.2 are transformed into the dependence of $\boldsymbol{x}$ and t, which are used in our case.

$$\boldsymbol{f}^i_{stiff,n+1} = \sum_{e\in\epsilon_r} \xi_e\boldsymbol{V}^T\boldsymbol{L}_e^T\boldsymbol{f}_e(\boldsymbol{L}_e\boldsymbol{x}^i_{n+1}, t_{n+1}) \tag{4.9}$$

$$\boldsymbol{K}^i_{n+1} = \sum_{e\in\epsilon_r} \xi_e\boldsymbol{V}^T\boldsymbol{L}_e^T\boldsymbol{K}_e(\boldsymbol{L}_e\boldsymbol{x}^i_{n+1}, t_{n+1})\boldsymbol{L}_e\boldsymbol{V} \tag{4.10}$$

4.1.4 Co-simulation approach

Here is a brief graphical representation of the procedure derived above. The point is to clearly summarize the structure of the code and the interactions between Python and Abaqus. The calculation of the subset ϵ_r and the corresponding weights for these elements-ξ is shown in the figure 4.1. A co-simulation procedure for the calculation with a hyper-reduced model is shown in the figure 4.2. The mass matrix $\boldsymbol{M}$ is assumed to be independent of the displacement and is calculated once at the beginning. $\boldsymbol{K}_e$ and $\boldsymbol{f}_{e,stiff}$ are calculated in each increment based on the displacement field $\boldsymbol{x}$ and time moment t received from Python. Subscribed e denotes the elemental values of the stiffness matrix and the nonlinear forces. Equations for $\boldsymbol{X}$ and $\boldsymbol{r}$ are derived in 3.32 and 3.31. Some further explanation is given in the pseudo-code in the following section (4.2).

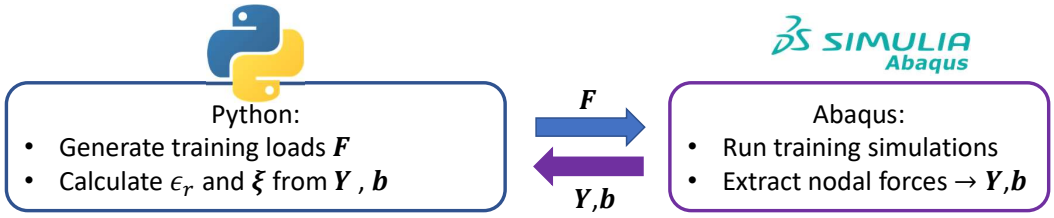


Figure (4.1) Calculation of the subset ϵ_r and weights ξ

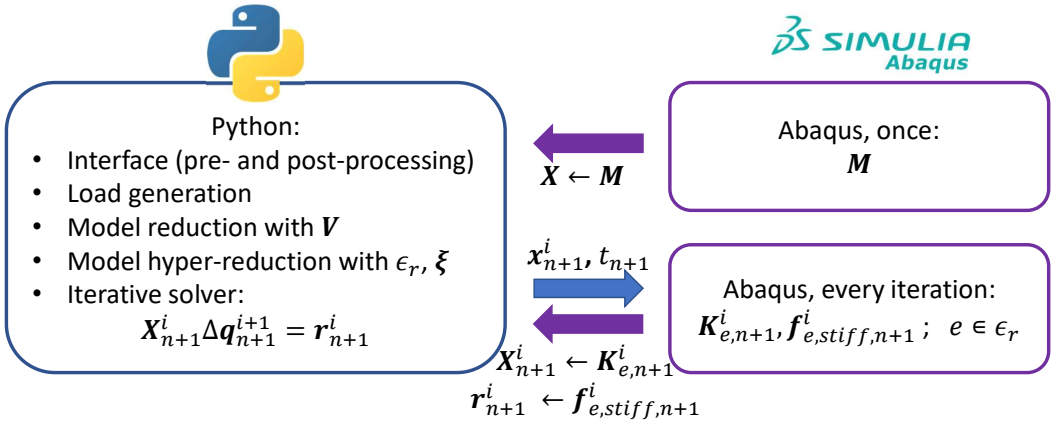


Figure (4.2) Co-simulation of the hyper-reduced model

4.2 Implementation

4.2.1 Calculation of the subset and weights

A rough overview of the procedure for calculating the subset of elements and the elemental weights is given in figure 4.1. A theory is explained in 4.1.2. Here, additional details that apply to our implementation case are described and a detailed pseudo-code is shown.

In general, several different training simulations are performed, with different values for amplitude, frequency and excitation node. The training simulations of the dynamic type, since time and inertia are important factors we want to capture. The loads are defined according to the definition in 2.2.6. Each training simulation is forced to be solved with 100 increments of equal length. Each time increment represents one training snapshot. The

number of the snapshots per training simulation is then $s = 100$. The data is collected in a similar way as for POD. Instead of snapshots of the displacement, snapshots of the nonlinear forces are collected here. The sNNLS method, which calculates subset and weights, will not be broken down further. It is considered as a standalone function with inputs: $\boldsymbol{Y}$, $\boldsymbol{b}$ and τ and outputs: ϵ_r and ξ. For further understanding, we refer to [17], where the function is presented through a pseudo-algorithm and some additional visualizations. The main workflow runs in Python. The steps that require interaction with Abaqus are indicated with a comment "// in Abaqus". More about Abaqus specific implementation is written in the appendix A.

Algorithm 3: Calculation of the subset ϵ_r and weights ξ

Input: *Ab_model* ... Abaqus model with defined mesh and material properties
LCs ... list of load cases. Load case is a dictionary with defined:
– f ... frequency of the load
– X ... amplitude of the load
– ψ ... set of nodes on which the load is applied
τ ... tolerance used in sNNLS

Output: ϵ_r ... set of selected elements
ξ ... vector of elemental weights

initialize $\boldsymbol{Y}$; // Empty matrix for collecting snapshots.
$ts \leftarrow$ length(LCs) ;
for $d \leftarrow 1$ **to** ts **do**
  $LC \leftarrow LCs$(d) ;
  Set up *Ab_model* with load LC ; // in Abaqus
  Run *Ab_model* simulation in Abaqus ; // in Abaqus
  Extract all nodal forces snapshots from Abaqus result file ; // in Abaqus
  Append forces to $\boldsymbol{Y}$; // equation 4.6
$\boldsymbol{b} \leftarrow \boldsymbol{Y} \cdot \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$; // sum up the rows of the matrix $\boldsymbol{Y}$
$(\epsilon_r, \xi) \leftarrow$sNNLS($\boldsymbol{Y}, \boldsymbol{b}, \tau$) ;
return ϵ_r, ξ

A remark on the possible time optimization. For better understanding, in this thesis the training simulations for the calculation of POD reduction and the ECSW hyper-reduction are performed in two separated loops (in two different chapters). However, as far the same training cases are used for POD and ECSW calculation, the trainings could be run once and the displacement fields and the nonlinear stiffness forces are extracted in parallel. This means all trainings are run once instead of twice.

4.2.2 Co-simulation of the hyper-reduced model

A rough overview of the procedure for calculation with a hyper-reduced model is given in the figure 4.2, and the theory is explained in 4.1.3. Additional details that apply to our implementation case are described here. The co-simulation is based on the procedure developed in 3.2 for simulating the (POD) reduced model. The only difference is that the stiffness matrix and the nonlinear forces are calculated for a subset of elements. For clearer understanding, an example of a displacement field applied to a subset is shown in the figure below, using subset of 10 elements. Other elements not present in the simulation are deactivated with an Abaqus interaction feature: "*MODEL CHANGE, REMOVE".

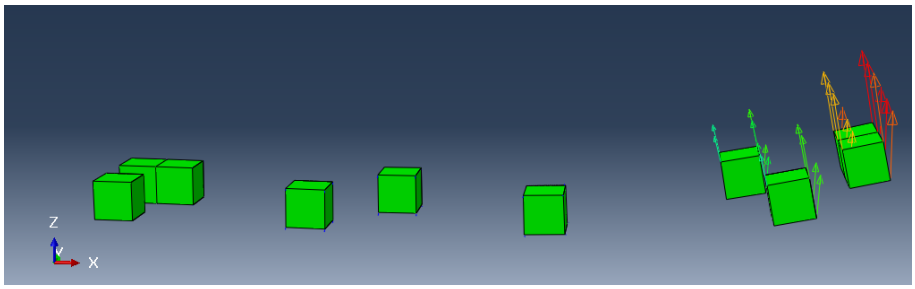


Figure (4.3) Displacement field applied on a subset of elements ϵ_r

Since the algorithm is almost the same as the one used for calculation with a (POD) reduced system (algorithm 2), it will not be shown again in full here. Rather, only two snippets of a few lines will be written, adapting the algorithm to make it suitable for calculation with a hyper-reduced model.

For the calculation of the stiffness matrix, line 24 in the algorithm 2 is replaced by the code:

Algorithm 4: ECSW calculation of stiffness matrix

```
1 K_{n+1}^{i+1} ← 0 ;                    // initialize stiffness matrix for new iteration
2 for e ∈ ε_r do                         // loop over subset of elements ε_r
3     K_{e,n+1}^{i+1} ← K_e(x_{n+1}^{i+1}, t_{n+1}) ;                    // in Abaqus
4 for e ∈ ε_r do                         // loop over subset of elements ε_r
5     K_{n+1}^{i+1}+ ← K_{n+1}^{i+1} + ξ_e P_e(K_{e,n+1}^{i+1}) ;        // in Python
```

In line 3, the elemental stiffness matrix is extracted from Abaqus, for a given time and deformation state. In line 5, the elemental contribution is weighted by ξ_e and localized with the mapping function P_e, which maps the elemental contributions to the corresponding locations in a global stiffness matrix. The code in the algorithm is executed in two separated loops, since in reality a first loop (extraction) runs in Abaqus, while the second is running in Python.

When calculating the nonlinear stiffness forces, line 25 in the algorithm 2 is replaced by the following code:

Algorithm 5: ECSW calculation of nonlinear stiffness forces

1 $f^{i+1}_{stiff,n+1} \leftarrow \mathbf{0}$; // initialize nonlinear forces vector for new iteration
2 **for** $e \in \epsilon_r$ **do** // loop over subset of elements ϵ_r
3 $\quad f^{i+1}_{e,stiff,n+1} \leftarrow f_{e,stiff}(x^{i+1}_{n+1}, t_{n+1})$; // in Abaqus
4 **for** $e \in \epsilon_r$ **do** // loop over subset of elements ϵ_r
5 $\quad f^{i+1}_{stiff,n+1} \leftarrow f^{i+1}_{stiff,n+1} + \xi_e P_e(f^{i+1}_{e,stiff,n+1})$; // in Python

In line 3, elemental nonlinear stiffness forces are extracted from Abaqus, for a given time and deformation state. In line 5, elemental contribution is weighted by ξ_e and localized with the mapping function P_e, that assigns the elemental contributions to the corresponding locations in a global vector.

Again, the main workflow runs in Python, while the steps performed in Abaqus are indicated with a comment "// in Abaqus". Some details about Abaqus implementation are given in the appendix A. The algorithm settings are the same as in 3.2. Each simulation is solved in 30 increments of equal length. The tolerance for a Newton-Raphson procedure is set to $1e-4$ and the maximum number of iterations per increment is set to 15. The loads are defined according to 2.2.6.

4.3 Training, Results, Discussion

Since the POD parameters settings were studied and explained in detail in the previous chapter, the optimal parameters are assumed as the starting point here. ECSW introduces an additional parameter that needs to be determined - τ. This parameter directly affects the number of selected elements in the reduced set ϵ_r. Here, the first subsection discusses the use of different values of τ, while the second subsection shows the calculation for the same general case as for POD.

4.3.1 Different value of τ

Training loads

The same training simulations are used here as for the general case in POD. They can be seen in the table 3.13 in the previous chapter.

Hyper-reduced model

Three different hyper-reduced models were built here. All three use $n = 16$ for the POD reduction. However, they differ in parameter τ - a threshold needed for calculation of the ECSW weights. The τ parameter was set so, that a subset of elements ϵ_r includes 10, 20 or 30 elements. The values of the τ are 0.55, 0.25 and 0.095.

Validation loads

A validation case was performed for all three models. The parameters are defined in the table below.

Vl. case	$f[Hz]$	$X[N]$	ψ
1	75	0.45	[135]

Table (4.1) Validation case for different value of τ

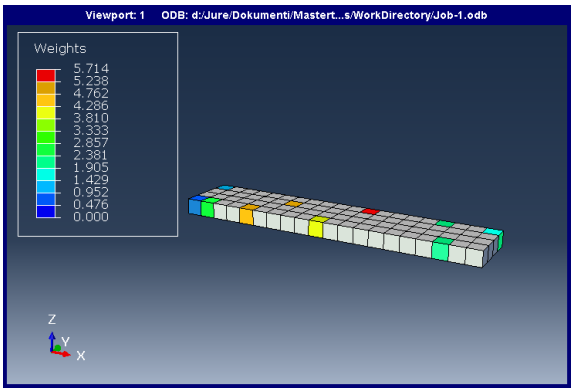
Results

First, the results of the calculation of the weights and the subset ϵ_r are presented. For the specified tolerances τ, the number of elements selected by algorithm is presented in the table 4.2. The number of elements in the subset ϵ_r is written as $|\epsilon_r|$.

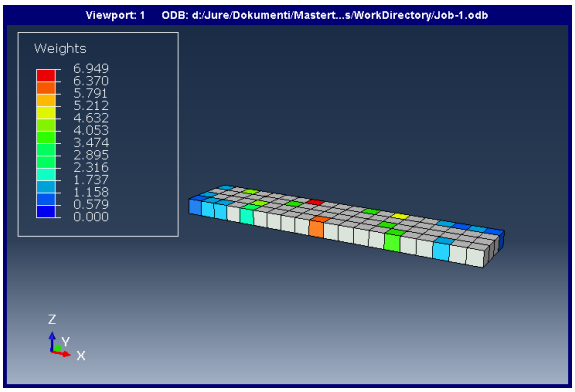
Hyper-reduced model	τ	$\lvert\epsilon_r\rvert$
1	0.55	10
2	0.25	20
3	0.095	30

Table (4.2) Co-relation between τ and number of elements in ϵ_r

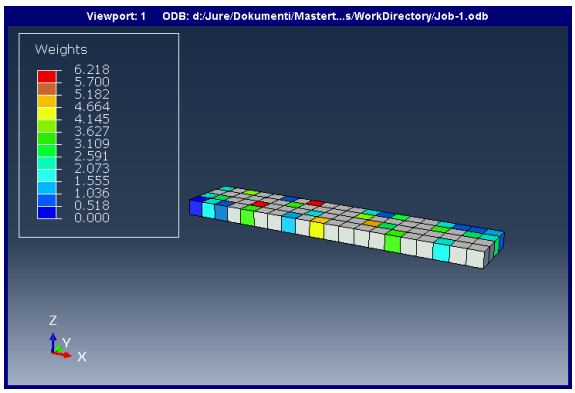
Selected elements with the corresponding values of the weights are depicted in the figures below.



(a) Hyper-reduced model 1



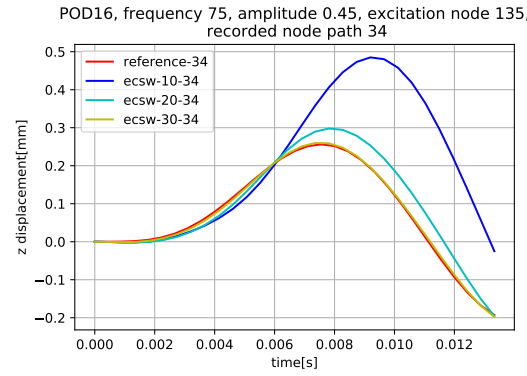
(b) Hyper-reduced model 2



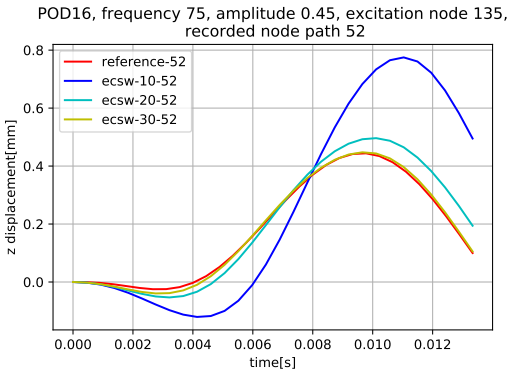
(c) Hyper-reduced model 3

Figure (4.4) Selected elements with corresponding weights

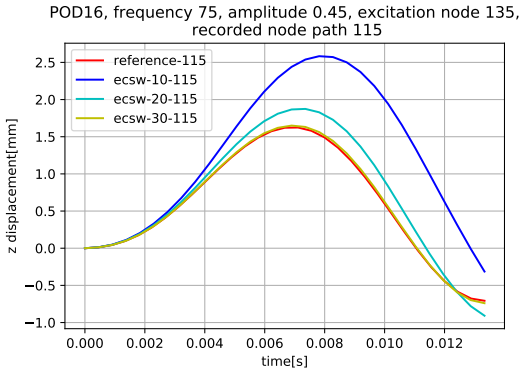
After calculating the weights and selecting the elements, the co-simulations of the hyper-reduced models were performed. The results are plotted in the figures bellow. Six nodes were recorded: 34, 52, 115, 152, 183 and 205. For each recorded node, the solutions of all three hyper-reduced models are plotted along with a reference solution calculated by Abaqus. The curves are named in the legend according to the number of elements used in a subset ϵ_r.



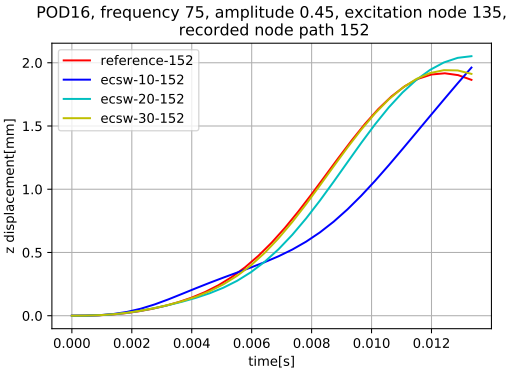
(a) Node 34



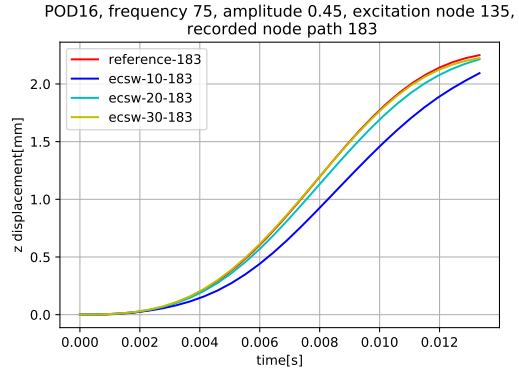
(b) Node52



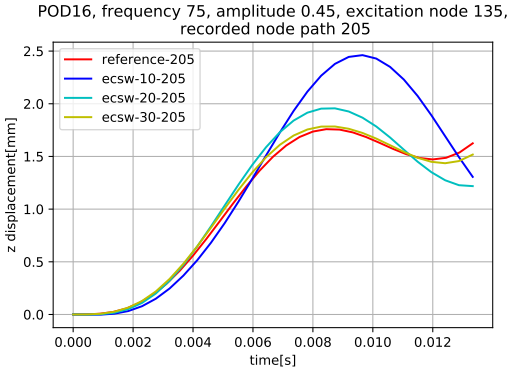
(c) Node 115



(d) Node152



(e) Node 183



(f) Node 205

Figure (4.5) Different value of τ

Discussion

It can be seen that a solution based on more elements generally means a better solution. This is also to be expected. Each reduction introduces a new error into the model in general. The greater the reduction, the greater the discrepancy between the reference and the reduced model. In our case, the solution with 30 elements agrees well with the reference solution, the solution with 20 elements is worse and solution with 10 elements even worse.

4.3.2 General example

Here, the generality of the fitted model will be proven. This will be done with the same training and validation cases as in the POD general example (3.3.5).

Training loads

Training cases for a general example can be seen in the table 3.13 in the previous chapter.

Hyper-reduced model

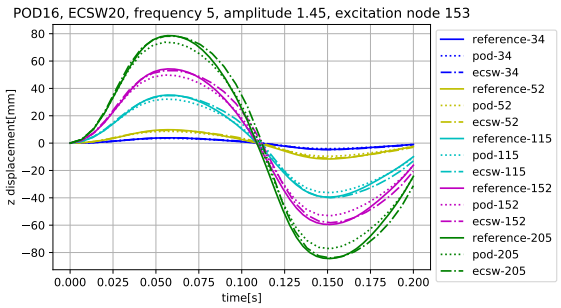
The hyper-reduced model created here uses $n = 16$ for a POD reduction and $\tau = 0.25$ for a ECSW weights calculation. $\tau = 0.25$ corresponds to 20 elements. The weights can be seen in the figure 4.4b.

Validation loads

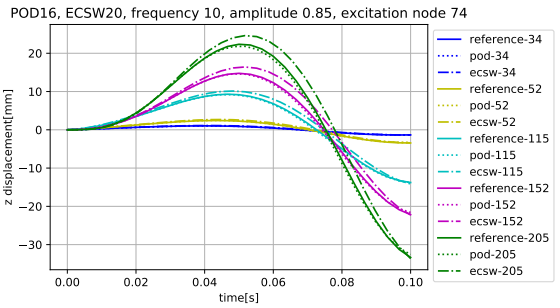
The same validation cases are used here as for the POD reduction. The validation cases for a general example can be seen in the previous chapter in the table 3.14.

Results

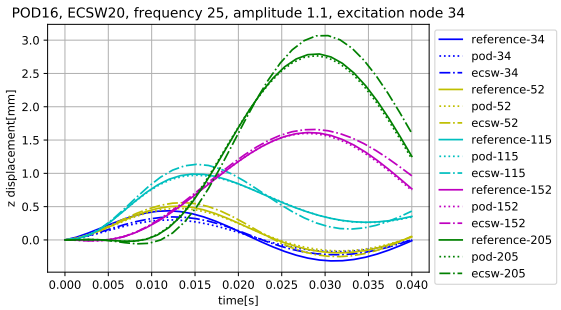
Here are the results for all 10 validation cases. Five nodes (34, 52, 115, 152 and 205) were recorded during the co-simulation. For each validation case, five nodes are plotted with a reference, a pod-reduced and an ecsw-hyper-reduced solution. The results of the POD reduced model are the same as already shown in the previous chapter (3.3.5).



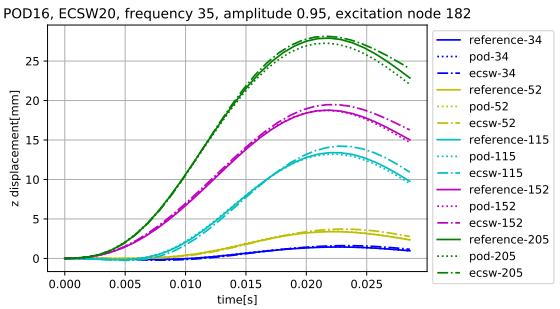
(a) Validation case 1



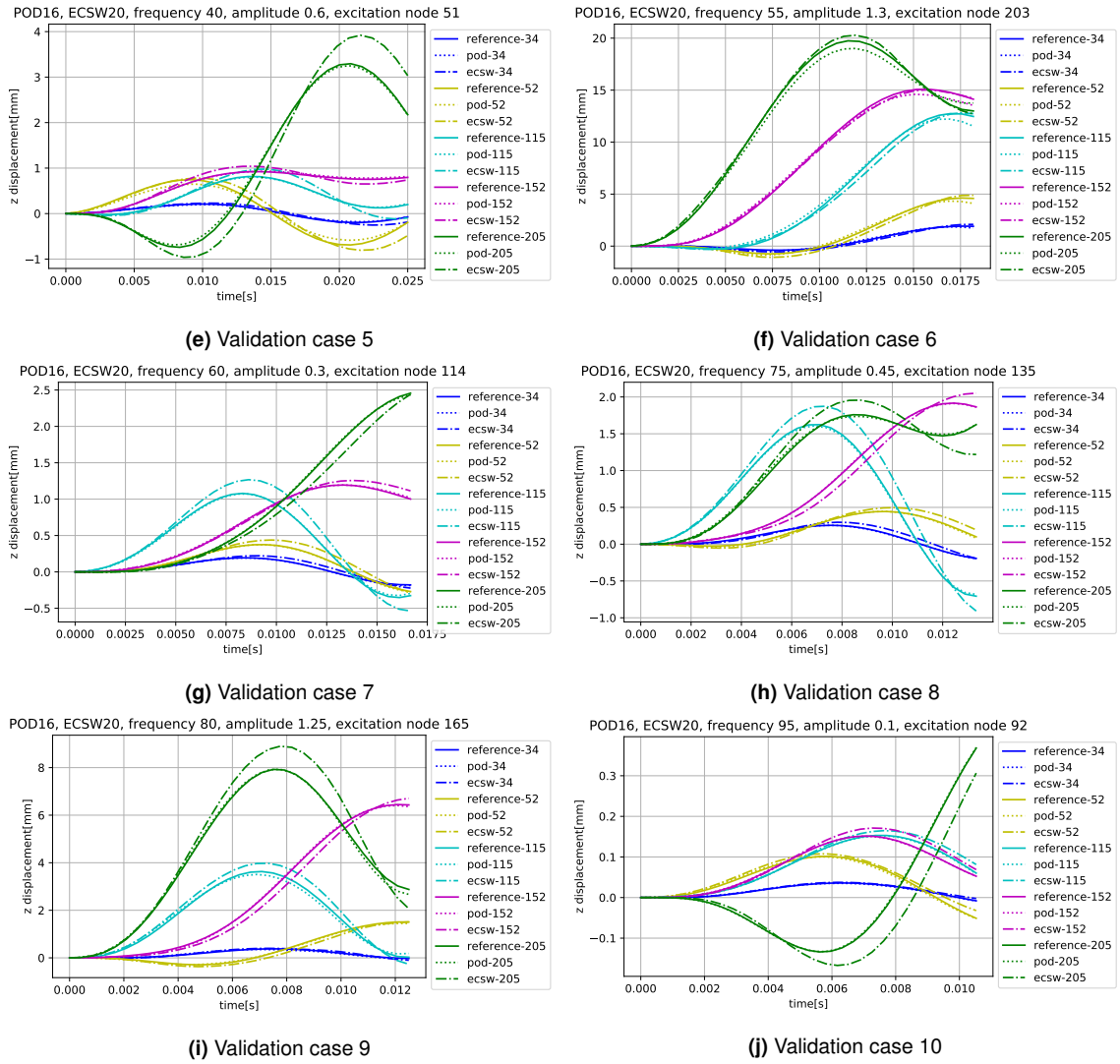
(b) Validation case 2



(c) Validation case 3



(d) Validation case 4



(e) Validation case 5

(f) Validation case 6

(g) Validation case 7

(h) Validation case 8

(i) Validation case 9

(j) Validation case 10

Figure (4.6) General validation cases - results

Discussion

The results are considered to be good. A co-simulation approach works in a sensible manner. It can be seen that the hyper-reduction of the model introduces an additional error into the system compared to the POD reduction only. However, in some specific cases, the error of the POD and the error of the ECSW approximation may cancel each other out and the result of ECSW ends up being better than that of the POD. Such a case can be seen in Validation case 1 in figure 4.6a.

The overall reduction is maybe not very impressive - reduction from 80 to 20 elements. However, it could be that the number of elements required for a quality approximation is driven by physics and is independent of the mesh. This would mean that for a certain spatial distribution of the selected elements (e.g., at about 20 different locations), the behavior would be well captured regardless of whether the bar contained 80 or 400 elements before hyper-reduction. A reduction from, say, 400 to 20 elements would then mean a much better reduction. This would be an interesting topic for further investigation.

4.4 Work-based ECSW calculation

In this section, we briefly present an alternative approach that gives better results for our example. This is not a complete argumentation with all mathematical derivations and proofs. It is just a note for someone who is willing to investigate this further.

4.4.1 Implementation

The concept is identical with everything presented in the chapter of ECSW (chapter 4) so far. The only difference is a definition of $\boldsymbol{Y}$ and $\boldsymbol{b}$. For this approach, the definitions of the equations 4.6 and 4.7 are changed. Let us start with a new definition of $\boldsymbol{Y}$.

The matrix $\boldsymbol{Y}$ has now the dimension $s \cdot ts \times |\epsilon|$. This means that for every snapshot (from each training simulation) and each element it stores one scalar value. (Previously it was of dimension $n \cdot s \cdot ts \times |\epsilon|$, because for each snapshot and each element it stored the vector of the nonlinear forces of the length n.) This scalar value is interpreted as the sum of all the work done on an element up to a given snapshot in a current training simulation.

$$\boldsymbol{Y} = \begin{bmatrix} W_{1,1} & \dots & W_{1,|\epsilon|} \\ \vdots & \ddots & \vdots \\ W_{s\cdot ts,1} & \dots & W_{s\cdot ts,|\epsilon|} \end{bmatrix} \tag{4.11}$$

The definition of scalar entries W for any $1 < p < s \cdot ts$ and $1 < q < |\epsilon|$.

$$W_{p,q} = \begin{cases} 0 & + \sum_{a \in \eta_q} \frac{\boldsymbol{f}_{p,q,a} + \boldsymbol{0}}{2} \cdot (\boldsymbol{x}_{p,a} - \boldsymbol{0}) & \text{if } mod(p/s) = 1 \\ W_{p-1,q} & + \sum_{a \in \eta_q} \frac{\boldsymbol{f}_{p,q,a} + \boldsymbol{f}_{p-1,q,a}}{2} \cdot (\boldsymbol{x}_{p,a} - \boldsymbol{x}_{p-1,a}) & \text{otherwise} \end{cases} \tag{4.12}$$

For easier understanding, the second term of the second line in equation 4.12 can also be rewritten as follows:

$$\sum_{a \in \eta_q} \frac{\boldsymbol{f}_{p,q,a} + \boldsymbol{f}_{p-1,q,a}}{2} \cdot (\boldsymbol{x}_{p,a} - \boldsymbol{x}_{p-1,a}) = \sum_{a \in \eta_q} \bar{\boldsymbol{f}}_{p,q,a} \cdot \boldsymbol{d}\boldsymbol{x}_{p,a} = \sum_{a \in \eta_q} \Delta W_{p,q,a} \tag{4.13}$$

The following diagram shows the procedure graphically. It shows the calculation for an element q, a node a and a snapshot p. A displacement of the node a is plotted on the x axis, and a force of the node a acting on the element q is plotted on the y axis. We approximate the integral of the force $\boldsymbol{f}_{q,a}$ with respect to the displacement $\boldsymbol{x}_a$ with a red area. This area represents the work. A small error is introduced because the upper boundary is approximated by a straight line for ease of calculation. However, the error is negligible if the snapshots are close together. The red surface could be then transformed to the green dashed one covering the same area. The green area is used to calculate the work, as written in 4.13.

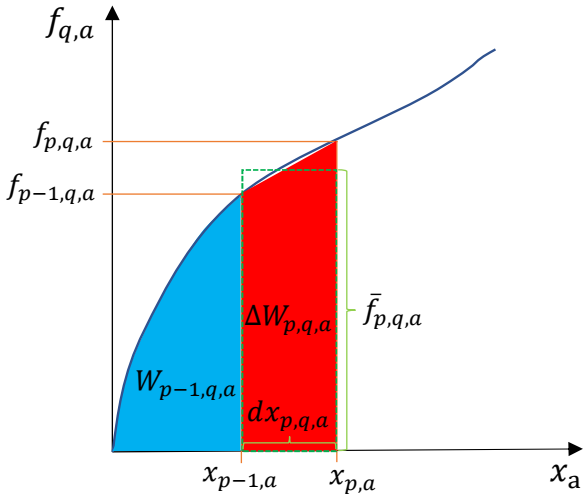


Figure (4.7) ECSW work-based calculation of $\boldsymbol{Y}$

The equations written above and the diagram require some further explanation and definition:

- η_q is a set of the nodes corresponding to the element q.
- a is an index of the node from the set η_q.
- $\boldsymbol{f}_{p,q,a}$ is a vector with 3 component (x,y,z) of the force acting in a snapshot p, on an element q and a node a.
- $\boldsymbol{x}_{p,a}$ is a vector with 3 components (x,y,z) of the displacement of the node a in a snapshot p.
- if $mod(p/s) = 1$ means in words "if the remainder of the p/s is equal to 1". Where s is the number of snapshots in one training simulation. The condition in other words reads: If the snapshot is the first in a training simulation, then start summing the work from zero. This is because it makes no sense to sum the work of different training simulations together.
- fraction of the forces $\boldsymbol{f}$ represents an average force $\bar{\boldsymbol{f}}$, acting between the snapshots p and $p-1$.
- the difference of the displacements $\boldsymbol{x}$ represents the step in the displacement dx, done between the snapshots p and $p-1$.
- The product of the average force $\bar{\boldsymbol{f}}$ and step of the displacement dx is a work produced on a node of an element between the snapshots p and $p-1$. It is indicated as $\Delta W_{p,q,a}$.
- Since a graph is plotted for only one node a, we are not able to show what $W_{p-1,q}$ directly is. However, $W_{p-1,q,a}$ is highlighted. It is a work done on a node a up to the previous snapshot. To be fully consistent the following relationship is written:

$$W_{p-1,q} = \sum_{a \in \eta_q} W_{p-1,q,a} \tag{4.14}$$

  In the code, $W_{p-1,q}$ is used directly without knowing $W_{p-1,q,a}$, since $W_{p-1,q}$ is a known entry of the $\boldsymbol{Y}$ for the previous snapshot.

- The first line of the equation 4.12 is for starting summing up the work of every training simulation. It begins from zero and the values of the "previous snapshot = starting position" are $\boldsymbol{f} = \boldsymbol{x} = \mathbf{0}$.
- The second line of the equation 4.12 subsequently adds the work done between two snapshots.

$\boldsymbol{b}$ is a vector of length $s \cdot ts$. It collects all the work produced on all elements for every snapshot and every training simulation.

$$\boldsymbol{b} = \begin{bmatrix} \sum_{e \in \epsilon} W_{1,e} \\ \vdots \\ \sum_{e \in \epsilon} W_{s \cdot ts,e} \end{bmatrix} \tag{4.15}$$

In practice, for sake of simplicity, it is written:

$$\boldsymbol{b} = \boldsymbol{Y} \cdot \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \tag{4.16}$$

4.4.2 Results

Let us define the standard ECSW calculation presented by [17] and introduced in 4.1.2 as method A and the work-based calculation presented in this section as method B. Method B performs exactly the same validation cases as Method A. Also, all settings are exactly the same, only the τ values are adjusted, so that we cane obtain 10, 20 or 30 elements as solution for both methods. With doing so, the reduced models are well comparable. The results of the method A have already been presented in the previous section 4.3. Here, the results for the method B are presented with the same headings: "Different value of τ" and "General example". In a "General example", a comparison between method A and method B is evaluated to make the difference clear.

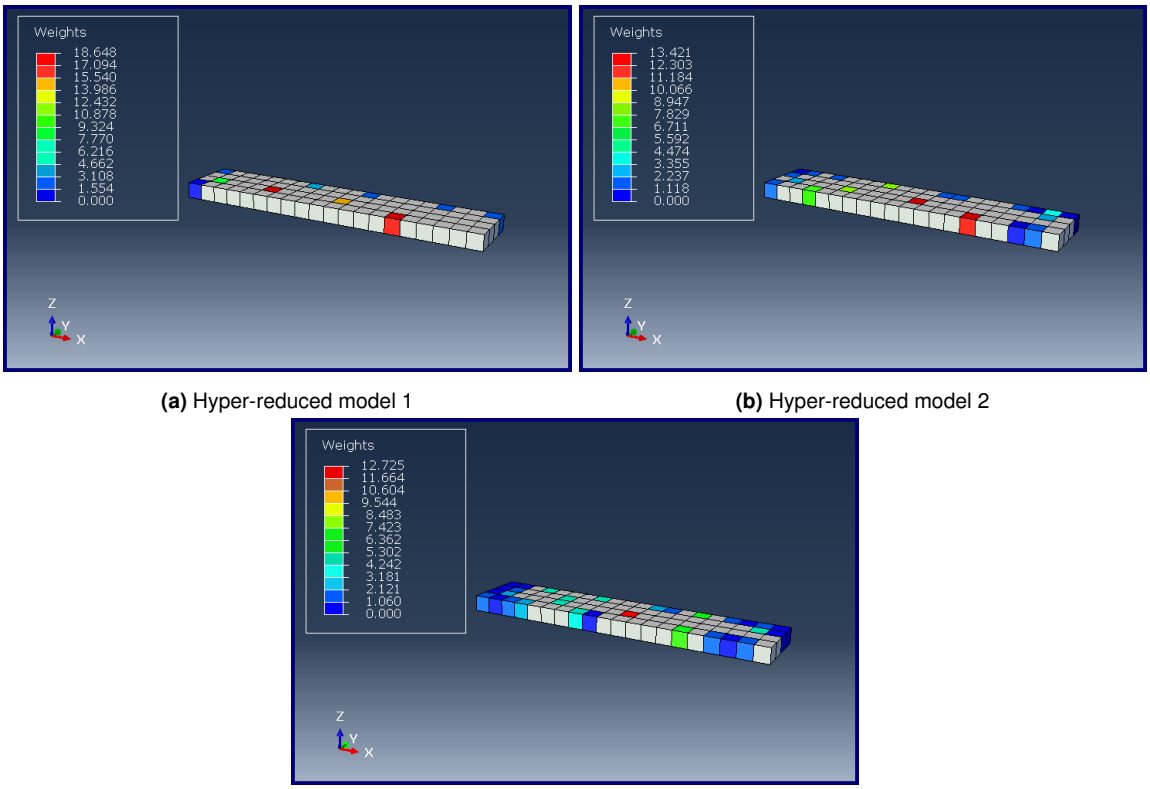
Different value of τ

The training and validation cases are the same as in 4.3.1. First, the results of calculation of the weights and subsets ϵ_r are presented. As already mentioned, the tolerance τ was assigned an appropriate value to obtain $|\epsilon_r|$ of 10, 20 or 30. For each τ, a reduced model is built.

Hyper-reduced model	τ	$\lvert\epsilon_r\rvert$
1	$8e-4$	10
2	$7e-5$	20
3	$1.3e-5$	30

Table (4.3) Co-relation between τ and number of elements in ϵ_r

Selected elements with the corresponding weights are depicted in the figures below.

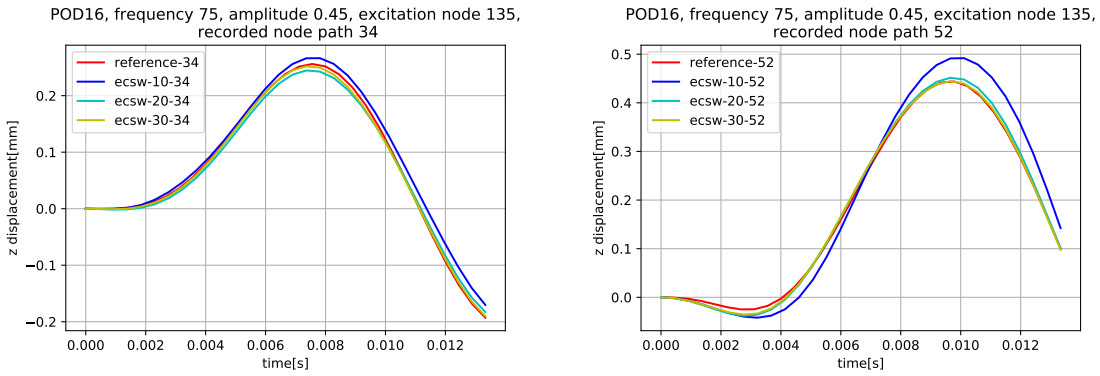


(a) Hyper-reduced model 1 **(b)** Hyper-reduced model 2

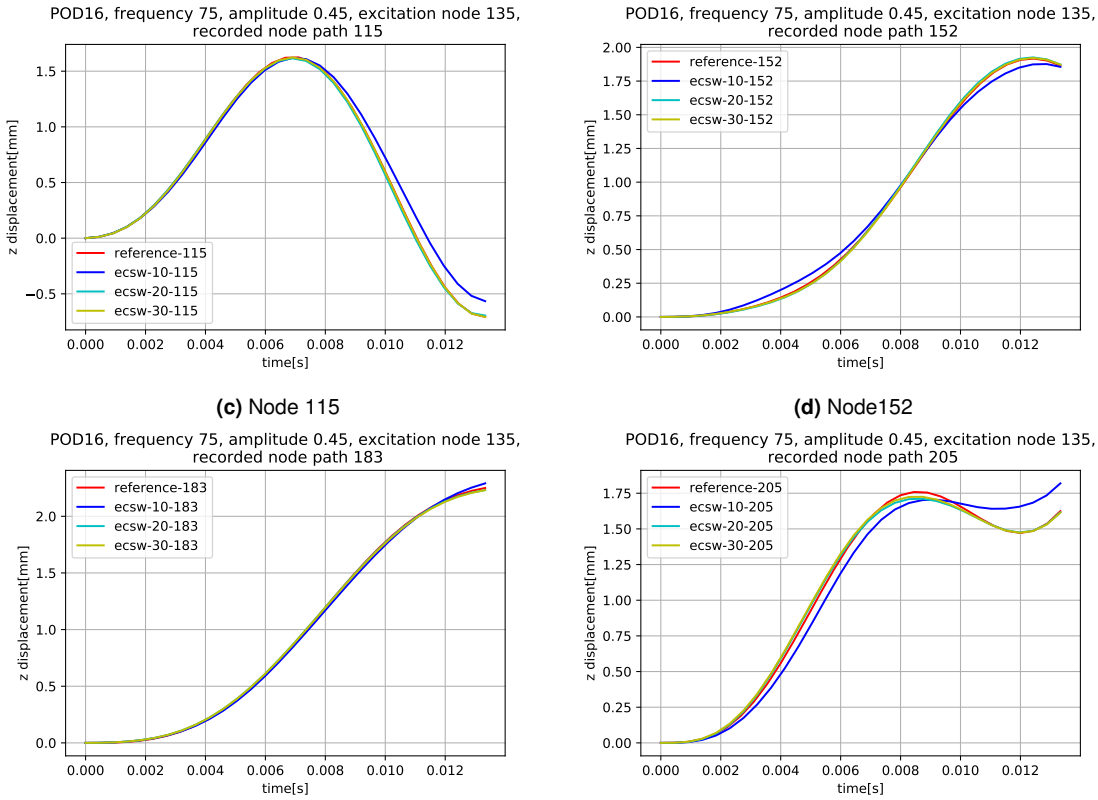
(c) Hyper-reduced model 3

Figure (4.8) Selected elements with corresponding weights

For each recorded node (34, 52, 115, 152, 183 and 205), a figure is shown on which the reference solution generated by Abaqus and the solutions of all three different hyper-reduced models are plotted. The curves in the legend are named after the number of elements used in a subset ϵ_r.



(a) Node 34 **(b)** Node52



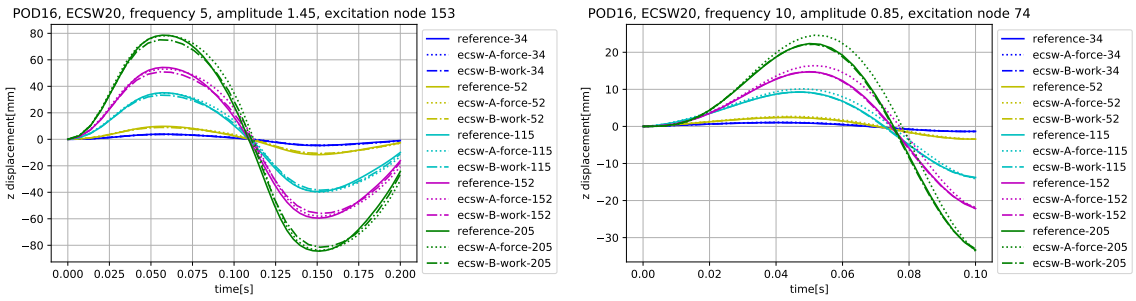
(c) Node 115 **(d)** Node152

(e) Node 183 **(f)** Node 205

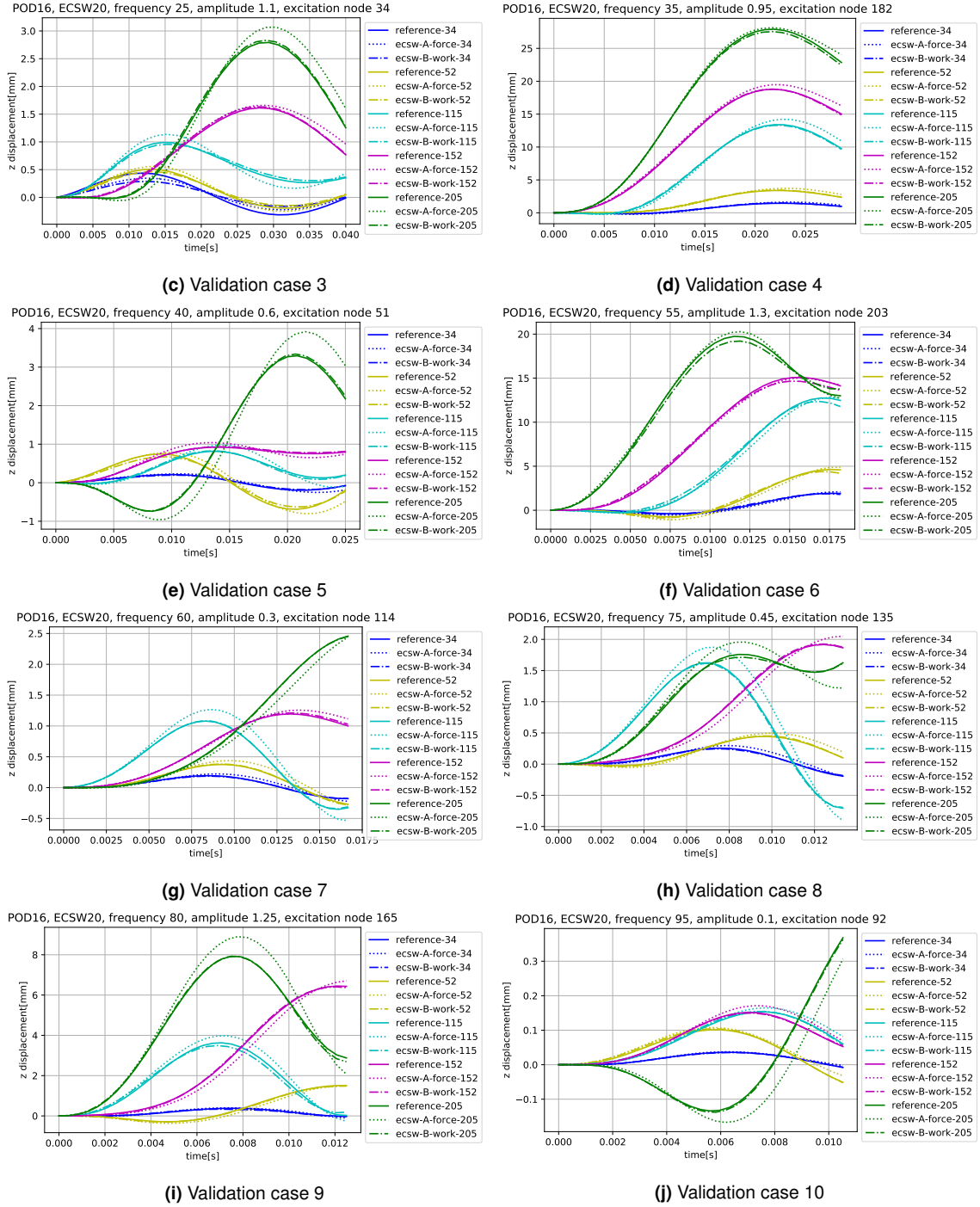
Figure (4.9) Different value of τ - Method B

General example

The general example uses exactly the same conditions as the general example presented in 4.3.2, where it was run with a method A. Here it will be run with a method B and τ value $7e-5$, which corresponds to $|\epsilon_r| = 20$. This results in a comparable reduction as with method A. In the figures below, the solution using method A is called ecsw-A-force, while that of method B is called ecsw-B-work. This is because method A uses the nonlinear forces in the $\boldsymbol{Y}$ matrix, while method B uses the work in the definition of $\boldsymbol{Y}$. The numbers in the legend represent the numbers of the plotted nodes.



(a) Validation case 1 **(b)** Validation case 2



(c) Validation case 3

(d) Validation case 4

(e) Validation case 5

(f) Validation case 6

(g) Validation case 7

(h) Validation case 8

(i) Validation case 9

(j) Validation case 10

Figure (4.10) General validation cases - Alternative approach

4.4.3 Discussion

From both results, it is clear that the Work-based approach - method B - gives much better results than the traditional approach - method A. However, this is true for our test scenario. Further studies on several different models would be required to generalize any statement. The results for two other models is presented in the appendix B.

A possible explanation for the difference between the methods could be that method A used

in [17] captures only nonlinear behavior (material or geometric). However, in our case, viscoelasticiy is also considered. Viscoelasticity implies a relevance of the loading history. In method A, the elements and weights are chosen independently of the loading history. Every snapshot is taken independently of all others. Method B, on the other hand, considers a history with the accumulation of the work done on each element during entire training simulation.

4.5 Time analysis

4.5.1 Analysis

In this section some time-related aspect are addressed. The time analysis was run on 3 different models, with the same validation case. It is Vl. case 6 from the table 3.14. Compared three models are introduced here:

1. **reference**
   The reference model is a full model set up in Abaqus.

2. **ecsw-20-elements**
   This is a model built with the training simulations for a general case presented in the table 3.13. It uses method B for evaluation of $\boldsymbol{Y}$ and $\boldsymbol{b}$. A threshold τ is set so, that 20 elements are chosen for an ecsw reduction.

3. **ecsw-80-elements**
   This is a model built with the training simulations for a general case presented in the table 3.13. It uses method B for evaluation of $\boldsymbol{Y}$ and $\boldsymbol{b}$. A threshold τ is set so, that 80 elements are chosen for an ecsw reduction. In an essence, no reduction is obtained here, since all 80 elements are used.

For model 1 (Abaqus reference model), only one time was measured. It is the total time needed to complete the simulation in Abaqus. The time is written in .dat file created by Abaqus itself. For models 2 and 3, a more detailed time measuring was performed. The scheme of the measured time is shown in the following figure:

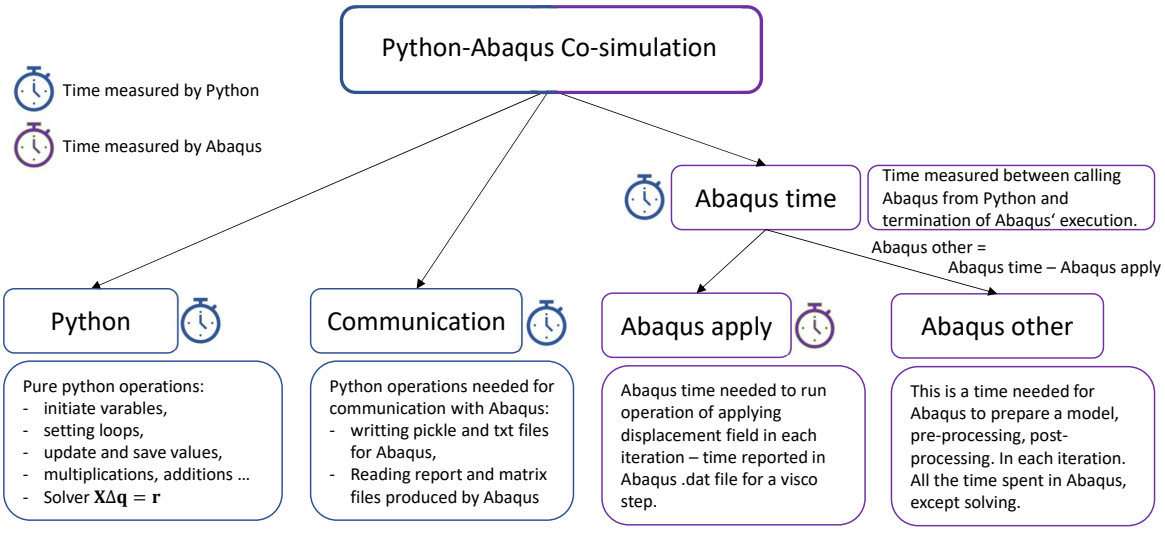


Figure (4.11) Scheme of the measured time for models 2 and 3.

4.5.2 Results

The results of the measured time are presented in the diagram below.

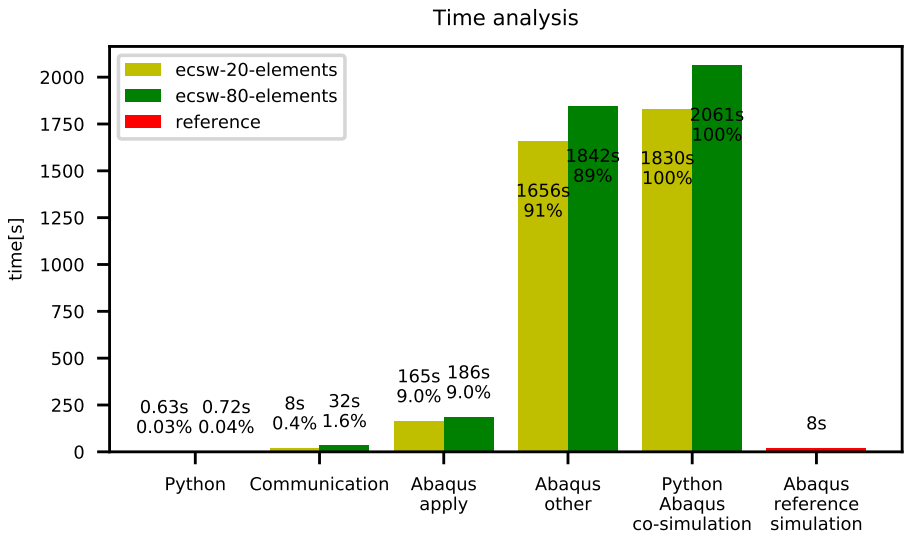


Figure (4.12) Time performance of the models 1, 2 and 3

4.5.3 Discussion

Some aspects of time analysis are summarized here:

- We will first discuss a comparison between the reference simulation time to the time required for co-simulation. With the current settings and environments, co-simulation takes several orders of magnitude more time than a simple Abaqus run. It is useful for academical proposes to see how the choice of different parameters affects a solution. One can also simply use complex formulations available in Abaqus to test in a relative sense the performances of the reduction strategy. However, at this stage, co-simulation is considered unsuitable for industrial use.
- The first focus of this thesis was methodology, not performance. In terms of performance, there is a lot of room for improvements. The fastest option would be for sure to write code inside of Abaqus, if that is possible. Otherwise, if we stay in Python-Abaqus co-simulation environment a radical improvement would likely be to write an .inp file directly from Python as a text for each iteration. In the current code, Abaqus is opened before each iteration, a model database is modified by a script and then Abaqus is acquired to write a new .inp file. This is slow, of course, because Abaqus has to be opened. Instead, the old .inp file could simply be opened in Python as a text file and the values to be changed overwritten. This should decrease Abaqus-other time.
- Scaling of the time with scaling of the model dimensions was not addressed in this thesis. This is certainly something that should be investigated.
- When using fewer elements in ecsw, the co-simulation takes less time. This behavior was expected. The scaling is not perfectly linear, but it is at least well recognizable. The only category that scales linearly is communication. Python spent much less time on reading and writing when fewer elements were used in a subset for ecsw.

Chapter 5

Conclusions and Outlook

5.1 Conclusions

The most important conclusions of this study are:

- It is possible to create a Python-Abaqus co-simulation environment for POD-ECSW hyper-reduced models consisting of hyper-viscoelastic material. A co-simulation was successfully performed, but it was relatively slow, at that point.
- The accuracy of the reduced model depends highly on the choice of training simulations and other reduction parameters.
- For quality POD reduction basis, the training simulations must be normalized - the displacement fields of the simulations are scaled to be of the same order of magnitude. The procedure is presented in 3.1.2.
- Amplitude and frequency dependence in reduced models could be successfully captured with training simulations covering a whole range of frequencies and amplitudes of interest.
- Nodes to which the loads of training simulations are applied must be spatially distributed throughout the model. In this case, the validation load could be applied to any node of the reduced model with good results.
- More basis vectors included in POD reduction basis, directly means a better approximation.
- More elements considered in ECSW subset ϵ_r, directly means a better approximation.
- Work based calculation of the weights (an alternative approach presented in 4.4) yields better results than force based calculation (presented in literature, e.g. [17]). However, the alternative approach has not been fully validated. Further proofs on different examples should be performed using this method.
- Python Abaqus Co-simulation is at that point some orders of magnitude slower than simple Abaqus dynamic analysis. The code, as is, is useful for academic proposes, to study the influence of various parameters or to test different complex formulations available in Abaqus. However, at this stage a co-simulation is considered unsuitable for industrial use. A code has some potential improvements that could reduce a time. They are explained in a discussion of the time analysis section (4.5.3).
- Co-simulation with fewer elements for ecsw, takes less time. The main relative time benefit of fewer elements is faster communication between Python and Abaqus.

5.2 Outlook

Several upgrades are possible on this project:

- Improvement of the code to work with hybrid elements (an additional pressure degree of freedom per element).
- Implementation of a more advanced time discretization method; e.g Newmark method.
- Further investigation of the Work-based ecsw. Generalization of this method must be demonstrated.
- Speed up of the co-simulation. The best way seems to be to write a code inside of Abaqus, if possible. In this case no time would be lost during opening and closing of Abaqus. In other case, at least an existing code could be optimized (e.g. writing the Job file directly from Python, without opening cae - see Time analysis discussion 4.5.3).
- Investigate the time scalability of the model - how the co-simulation time is changing with a dimension of the model.
- Try to apply hyper-reduction only to a nonlinear part of the stiffness forces. Possibly this would reduce the error.
  In this thesis, ECSW reduces a whole right hand side of the equation 5.1. The proposal is to use ECSW to approximate only the nonlinear part as represented in 5.2 in a term $\boldsymbol{f}_{nlin}$. The linear part remains unchanged. Consequently, no error is introduced into the linear part. $\boldsymbol{K}_{lin}$ is a linearized stiffness matrix in a reference position.

$$\boldsymbol{f}_{stiff} = \boldsymbol{f}_{stiff}(\boldsymbol{x}, t) \tag{5.1}$$

$$\boldsymbol{f}_{stiff} = \boldsymbol{K}_{lin}\boldsymbol{x} + \boldsymbol{f}_{nlin}(\boldsymbol{x}, t) \tag{5.2}$$

Appendix A

Insight into Abaqus

In the appendix some Abaqus related things are presented. It briefly describes some important settings and commands that enable the necessary communication. This is more an overview than a detailed implementation guide. For further explanations the reader is referred to the Abaqus documentation.

A.1 Running Abaqus from Python

Let us call a main program script "main_script.py". In our case this script is written in Python 3.7 environment. In the code, three different commands are used to run Abaqus directly from "main_script.py". All three use the Python module "os" and in this environment a function "system". With os.system the user has access to a command prompt.

1. os.system('abaqus cae script=script_example.py')
   This command call Abaqus to open a CAE window (Abaqus' user interface) and execute all the commands written in "script_example.py". After all commands are executed, Abaqus remains open and is waiting for user interaction. All commands in the script have to be written in Python 2.7, which is a version that Abaqus uses. Abaqus has its own Python environment which is completely independent from any other environment on the computer. Almost all available commands of CAE are scriptable. Therefore, script can be used for editing the .cae model, running a job and for extracting the results from the .odb file of Abaqus.
   This command is particularly useful for debugging to see what is happening with a model. Once the code is working, it can usually be replaced with the following command.

2. os.system('abaqus cae noGUI=script_example.py')
   This command opens Abaqus in the background and works very similar to command 1. There are two differences from command 1. The first is that the Abaqus window is not visible here. The second is that when all commands in script_example.py finished, Abauqs is closed and main_script.py continues to run.

3. os.system('abaqus job=input_example interactive')
   This command is different from the previous two. It runs an Abaqus solver in the background. The information for the solver is passed through input file (e.g. "input_example.inp"). Input file must be written in the Abaqus input format.

A.2 Abaqus Input/Output precision

This is an important point for a convergent solutions. Some facts about Abaqus precision:

- The precision of Abaqus report file, where displacement fields and nonlinear forces are written, can be set manually. One can use up to 9 decimal places of precision. In our case all 9 decimal places were used.
- The precision of the boundary condition applied inside of CAE is reduced to 6 decimal places when it is written to the input file, regardless of the number of decimal places used to define it in CAE.
- The Abaqus solver can accept at least 9 decimals. Maybe even more, but 9 were checked in a comparison with the output, which reports the results with 9 decimals precision. Precision of 9 decimal places was sufficient for our implementation.

A.3 Applying a displacement field

A displacement field must be applied in each iteration of the algorithm. Then, the nonlinear forces as well as the stiffness matrix of the deformed configuration can be extracted. There are several options in which the displacement field could be applied.

Assigning of a displacement
The only way to deform a model to a desired deformation state is to apply a desired displacement filed as a boundary condition. In Abaqus, there are two ways to do this:

- Use *CAE* (this is the user interface of Abaqus) and assign the displacement field as a discrete field and then use it as a boundary condition. The problem of this method is that when Abaqus writes the input file, a precision of only 6 decimal places is written to the boundary condition, regardless of the precision used in CAE.
- Use *Input file*. Write boundary condition dof (degree of freedom) per dof directly into an input file. The input file looks something like this:

```
*Boundary
beam-1.1, 1, 1, -0.0294227687222
beam-1.1, 2, 2, 0.00780142625964
beam-1.1, 3, 3, 0.0329765402672
beam-1.2, 1, 1, -0.0250611663439
beam-1.2, 2, 2, 0.00515050792703
beam-1.2, 3, 3, 0.0198880132247
beam-1.3, 1, 1, -0.0190819734005
beam-1.3, 2, 2, 0.00272078362617
beam-1.3, 3, 3, 0.0129199870116
```

Figure (A.1) BC in input file

  In this case, the solver takes at least 9 decimal places (Maybe even more, but 9 were checked in a comparison with the output, which reports the results with 9 decimals precision.)

This is a very important detail that can influence the convergence. In some cases, convergence is not achieved if only 6 decimal places are used. Therefore, our code uses the

second approach - writing the boundary conditions directly into the input file.

Step
The displacement boundary condition with nonzero values could only be applied in Abaqus with a suitable step. It is worth to mention here, that since the boundary condition is applied to the entire domain of the model, there is no degree of freedom left. A solver only applies the displacements and it converges ideally in one iteration. Although a dynamic co-simulation is performed (inertia is important), from Abaqus the pure stiffness forces are acquired and therefore a static or quasi-static step is required.

- *STATIC is a command of the input file for a static step. It means that all time dependencies are neglected. This type of the step is appropriate if non-time dependent material is used.
- *VISCO is a command of the input file for a visco step in Abaqus. In other words this is a quasi static step. This means that the inertia is still neglected, while the time dependence of the material is taken into account. This was used in our case because of the viscoelastic material properties.

Restart function
A command: "*Restart, write, number interval=1" could be placed in the input file in a step definition in output requests section. This instruct Abaqus to write a restart file at the end of the step. From this restart file further analysis can then be started. In the restart file all required information is written (displacement, time, material state - very important for viscoelastic material).

In the case of *non-time dependent material*, it is not necessary to start each next increment with a restart function from a previous increment. Since stiffness does not depend on time, we could also apply displacement in each iteration starting from the initial state. The forces will always be the same, when a specified deformation state is reached. However, using a restart function and applying the displacement in smaller steps could improve the numerical convergence of the solution even in the case of non-time dependent material.

In case of *time dependent material* it is necessary to use the restart function. Here a history of the material is important. The concept of the Python solver we developed is that each iteration starts from the restart file of the last converged increment, since this is the last valid state of the material. (The iterations are just the guesses with some residual and therefore it does not make sense to continue from there.) Once the iteration converges, a new restart file is written and a new increment begins. An algorithm is written below. *inc_num* in the algorithm represents the total number of increments.

Algorithm 6: Structure of restarting

```
write restart(inc = 0) ;                    // write a restart for initial state
for inc ← 1 to inc_num do                   // incrementation loop - Backward Euler
    while iterating; iteration i do         // iteration loop - Newton
        From restart; Abaqus(x_inc^i, t_inc) ;
        Calculate x_inc^{i+1} ;             // see procedure in algorithm 2
    write restart(inc) ;                    // write a restart for converged increment
```

A.4 Exporting displacement field

After performing an analysis in Abaqus, the Abaqus is run with a second command presented in A.1. Initially, an .odb result file is opened. A snippet of the code can be seen in the figure below.

```
1  # open odb
2  o = session.openOdb(name='Job-1.odb')
3  session.viewports['Viewport: 1'].setValues(displayedObject=o)
4  session.viewports['Viewport: 1'].makeCurrent()
5
6  # format settings
7  nf = NumberFormat(numDigits=9, precision=0, format=ENGINEERING)  # set 9 decimal precision
8  session.fieldReportOptions.setValues(numberFormat=nf)
9  session.fieldReportOptions.setValues(reportFormat=COMMA_SEPARATED_VALUES)
10
11 # write displacements in ab_displacements.rpt file
12 session.writeFieldReport(fileName='ab_displacements.rpt', append=OFF,
13     sortItem='Node Label', odb=o, step=0, frame=1, outputPosition=NODAL,
14     variable=(('U', NODAL, ((COMPONENT, 'U1'), (COMPONENT, 'U2'), (COMPONENT, 'U3'), )), ), stepFrame=ALL)
15
```

Figure (A.2) Code for extraction of displacement field

A.5 Exporting nodal forces

Before running an analysis, a variable NFORC must be requested as a field output in an input file. This is a variable of the nodal forces. This is done by input file commands:

```
*Output, field
*Node Output
U
*Element Output, directions=YES
NFORC
```

Figure (A.3) Command for saving of nodal forces into .odb file

When the analysis is complete, a result file is opened in Abaqus. It is important that a **non-averaged field output** result is set. This is done in line 7 in the A.4 figure. The output precision is again set to 9 decimals.

- In *averaged field output result*, each node has one vector force, which is calculated as the average of all contributions from the surrounding elements. It is always zero (because the sum of forces on a node is zero), except when an external force is applied on the node. These forces are not of our interest.

- We are interested in a *non-averaged field output result* where for each node there are as many forces as there are many elements sharing that node. Thus, each element generates its own force with its isolated deformation. These are the forces we are interested in.

```python
# open odb
o = session.openOdb(name='Job-{}_matrix.odb'.format(step_num))
session.viewports['Viewport: 1'].setValues(displayedObject=o)
session.viewports['Viewport: 1'].makeCurrent()

# format settings
session.viewports['Viewport: 1'].odbDisplay.basicOptions.setValues(averageElementOutput=False)
nf = NumberFormat(numDigits=9, precision=0, format=ENGINEERING)
session.fieldReportOptions.setValues(numberFormat=nf)
session.fieldReportOptions.setValues(reportFormat=COMMA_SEPARATED_VALUES)

# write disassembled (elemental) nodal forces in ab_stiff_forces.rpt file
session.writeFieldReport(fileName='ab_stiff_forces.rpt', append=OFF,
        sortItem='Element Label', odb=o, step=0, frame=1, outputPosition=ELEMENT_NODAL,
        variable=(('NFORC1', ELEMENT_NODAL), ('NFORC2', ELEMENT_NODAL), ('NFORC3', ELEMENT_NODAL), ), stepFrame=ALL)
```

Figure (A.4) Code for extraction of nonlinear forces

A.6 Exporting mass and stiffness matrix

Mass and stiffness matrices are exported from Abaqus in a special step called "matrix generate". This step could can only be assigned via command in the input file. It is not supported in the cae. The step falls into the group of Abaqus' general steps (for more information see Abaqus documentation), which assumes to continue an analysis from the last general step. Both the static and visco steps (two used in this thesis) are general steps. Thus, running the matrix generate step after static or visco with an applied deformation, returns the values of the new, deformed and nonlinear dependent configuration. In our case, the matrix generate step is always placed after the visco step as just described. An example is shown in the figure below.

```
*Step
*Matrix generate, mass, stiffness
*Matrix output, mass, stiffness, format=labels
*End Step
```

Figure (A.5) Matrix generate step for mass and stiffness matrix

Additional option has to be considered, when exporting stiffness matrix for ECSW procedure, where the stiffness matrix must be reported disassembled (not assembled in a global matrix). This is the Abaqus option ELEMENT BY ELEMENT. An example is shown in the figure below.

```
*Step
*Matrix generate, stiffness, element by element
*Matrix output, stiffness, format=labels
*End Step
```

Figure (A.6) Matrix generate step for disassembled stiffness matrix

A.7 Poisson's ratio limitation

Rubberlike materials usually have very high Poisson's ratios (often higher than 0.495). However, Abaqus does not allow you to perform an analysis with a Poisson's ratio higher than 0.495 without using of hybrid elements (an error is returned). Hybrid elements are elements with an additional pressure degree of freedom. They aim to avoid convergence problems in simulations with nearly incompressible materials. If those elements would be involved in analyses the calculating procedure would need to be adjusted, since additional dofs with additional physical relations would be introduced. In our case, a Poisson's ratio of

0.495 (or an equivalent value of D_1 in a hyperelastic material - see 2.2.2) was used and there was no need to use hybrid elements.

Appendix B

Other results on Work-based ECSW calculation

Here, the original model used in this thesis was modified in two different configurations with different underlying dynamics. The purpose is to perform a quick validation to see if the Work-based ECSW calculation gives better results only as a lucky scenario for one configuration, or if it is valid for some more cases.

B.1 Configuration of a half length

In this configuration, the model presented in 2.2.1 was shortened. The dimension a from a figure 2.5 was changed form $100mm$ to $50mm$. The mesh was adjusted so that the elements were also shortened (figure B.1). Consequently, the node numbering introduced in the thesis remains valid (figure 2.6).

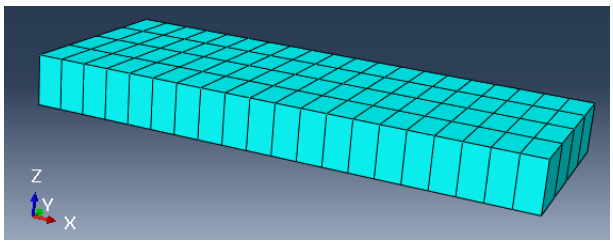
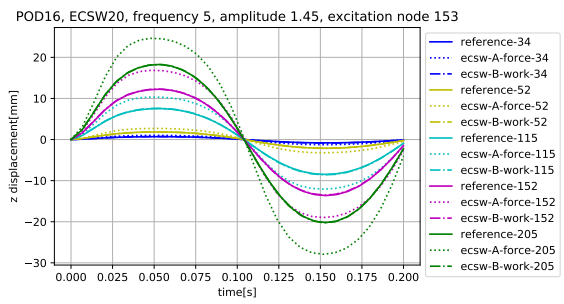
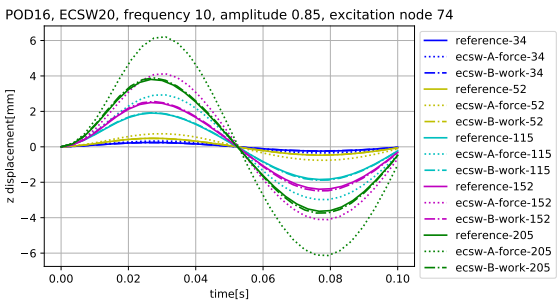


Figure (B.1) Beam of a half length

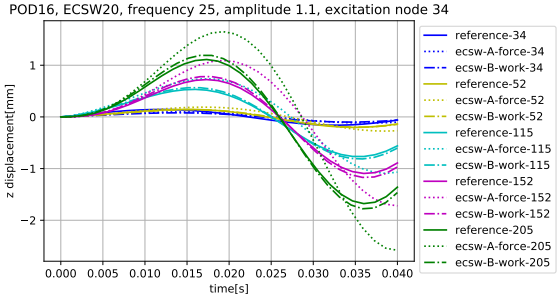
The trainings and validations are the same as in the general example (4.3.2). POD reduction with $n = 16$ and hyper-reduction with 20 elements were used. The results for method A-force (traditional way presented in [17]) and method B-work (alternative approach presented in 4.4) are shown in parallel in the figures below. Five nodes (34, 52, 115, 152 and 205) were recorded.



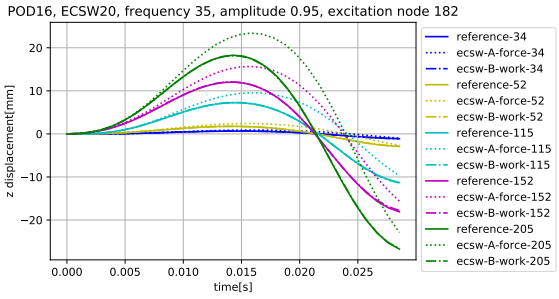
(a) Validation case 1



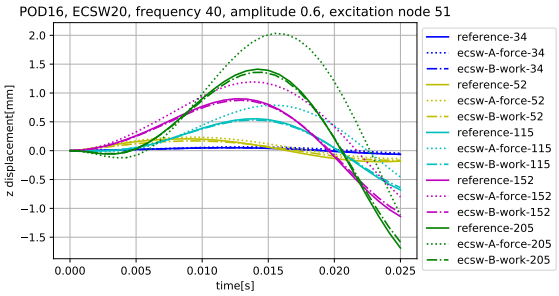
(b) Validation case 2



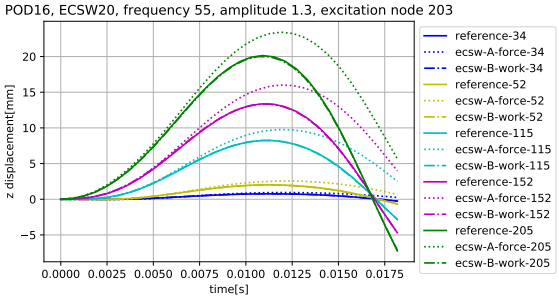
(c) Validation case 3



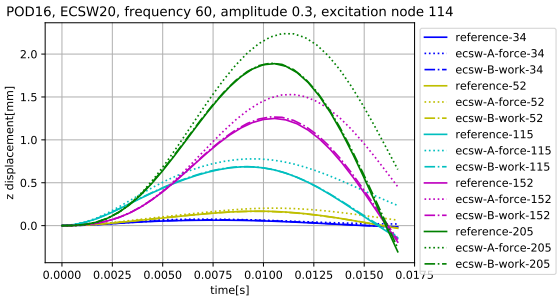
(d) Validation case 4



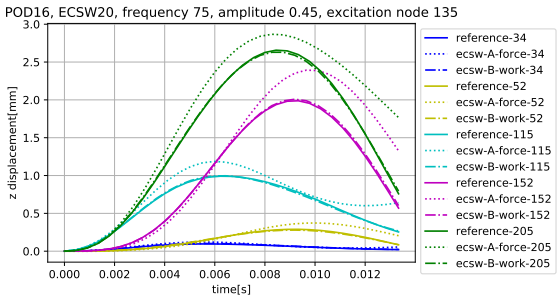
(e) Validation case 5



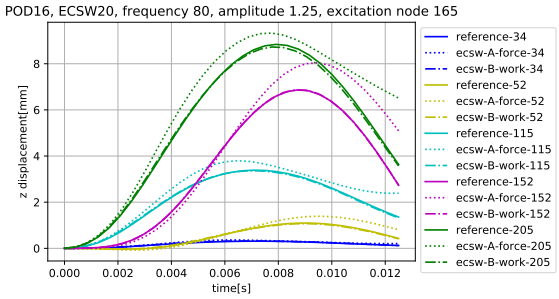
(f) Validation case 6



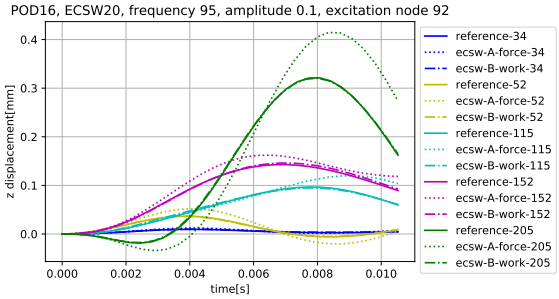
(g) Validation case 7



(h) Validation case 8



(i) Validation case 9



(j) Validation case 10

Figure (B.2) Half-length configuration results

B.2 Configuration of both-sides clamped model

Here the model presented in 2.2.1 was clamped on both sides. An additional boundary condition was set to the right end of the beam (figure B.3). The mesh was retained. Consequently, the node numbering introduced in the thesis remains valid (figure 2.6).

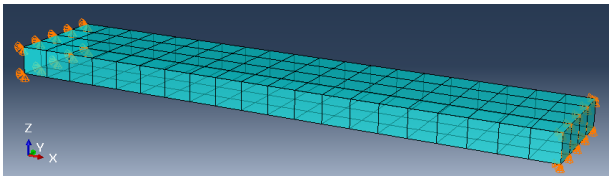


Figure (B.3) Beam clamped on both sides

Trainings and validations have changed, because the load is not allowed to be placed on the right end of the beam. In such a case, the load and the boundary condition would conflict. Training and validation cases are listed in the tables B.1 and B.2. POD reduction with $n = 16$ and hyper-reduction with 20 elements were used.

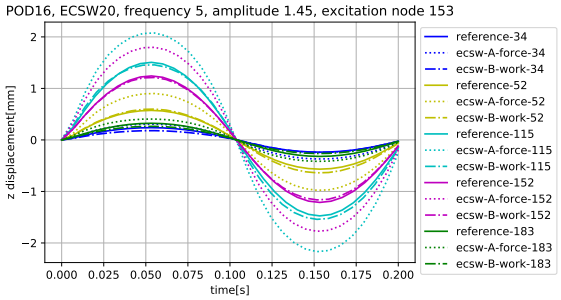
Tr. case	$f[Hz]$	$X[N]$	ψ
1	1	0.1	[21]
2	1	0.1	[182]
3	61	0.1	[64]
4	61	0.1	[142]
5	81	0.1	[25]
6	81	0.1	[92]
7	101	0.1	[25]
8	101	0.1	[181]
9	1	0.8	[64]
10	1	0.8	[125]
11	21	0.8	[92]
12	21	0.8	[175]
13	41	0.8	[64]
14	41	0.8	[175]
15	81	0.8	[125]
16	81	0.8	[165]
17	21	1.5	[21]
18	21	1.5	[92]
19	41	1.5	[64]
20	41	1.5	[175]
21	61	1.5	[51]
22	61	1.5	[142]
23	101	1.5	[34]
24	101	1.5	[181]

Table (B.1) General training loads

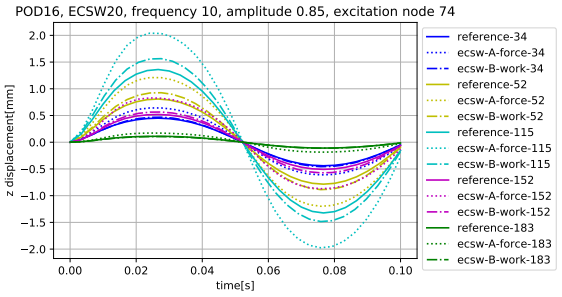
Vl. case	$f[Hz]$	$X[N]$	ψ
1	5	1.45	[153]
2	10	0.85	[74]
3	25	1.1	[34]
4	35	0.95	[182]
5	40	0.6	[51]
6	60	0.3	[114]
7	75	0.45	[135]
8	80	1.25	[165]
9	95	0.1	[92]

Table (B.2) General validation loads

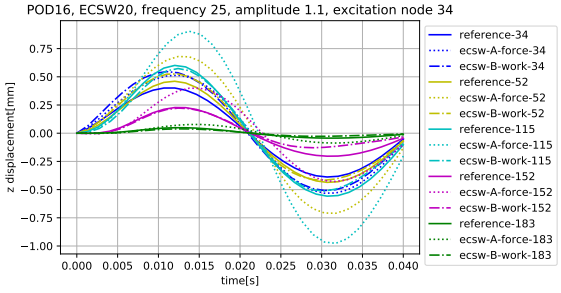
The results for the method A-force and method B-work are shown in parallel in the figures below. Five nodes (34, 52, 115, 152 and 183) were recorded.



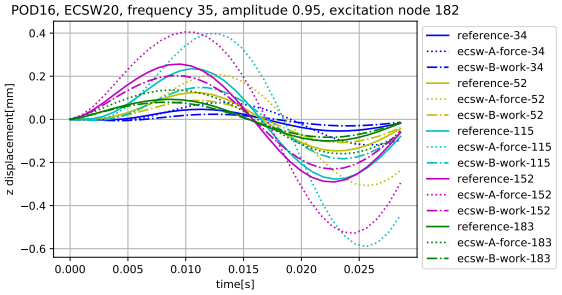
(a) Validation case 1



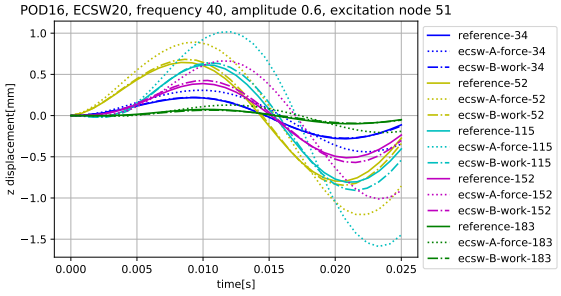
(b) Validation case 2



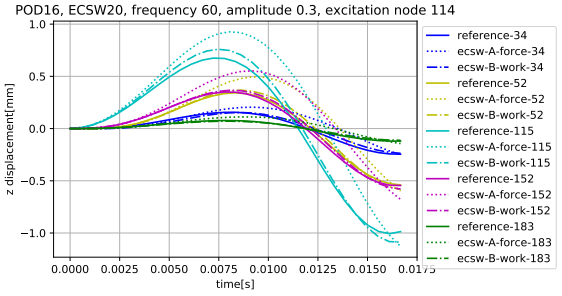
(c) Validation case 3



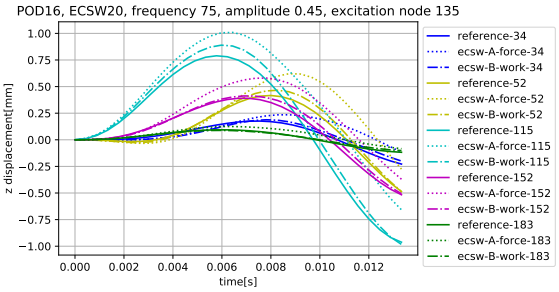
(d) Validation case 4



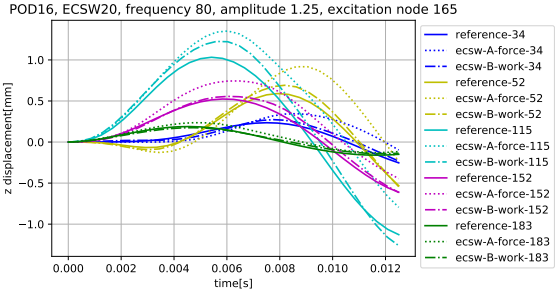
(e) Validation case 5



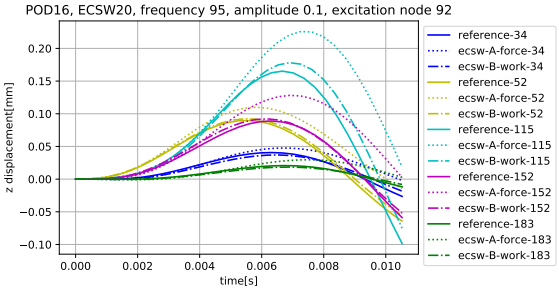
(f) Validation case 6



(g) Validation case 7



(h) Validation case 8



(i) Validation case 9

Figure (B.4) Results of both sides clamped beam configuration

Bibliography

[1] Ahmadi, H., Kingston, J., and Muhr, A. "Dynamic Properties of Filled Rubber — Part I: Simple Model, Experimental Data and Simulated Results". In: *Rubber Chemistry and Technology* 81 (Mar. 2008), pp. 1–18. DOI: 10.5254/1.3548196.

[2] Allen, M. S., Rixen, D., Seijs, M. van der, Tiso, P., Abrahamsson, T., and Mayes, R. L. *Substructuring in Engineering Dynamics*. Springer International Publishing, 2020.

[3] An, S. S., Kim, T., and James, D. L. "Optimizing Cubature for Efficient Integration of Subspace Deformations". In: *ACM Trans. Graph.* (2008).

[4] Austrell, P. E. "Modeling of Elasticity and Damping for Filled Elastomers". eng. PhD thesis. Lund University, 1997.

[5] Berkooz, G., Holmes, P., and Lumley, J. L. "The Proper Orthogonal Decomposition in the Analysis of Turbulent Flows". In: *Annual Review of Fluid Mechanics* 25.1 (1993), pp. 539–575.

[6] Chaturantabut, S. and Sorensen, D. C. "Nonlinear Model Reduction via Discrete Empirical Interpolation". In: *SIAM J. Sci. Comput.* 32 (2010), pp. 2737–2764.

[7] Craig, R. R. and Bampton, M. C. C. "Coupling of substructures for dynamic analyses". In: *AIAA Journal* 6.7 (1968), pp. 1313–1319.

[8] Guyan, R. J. "Reduction of stiffness and mass matrices". In: *AIAA Journal* 3 (1965), pp. 380–380.

[9] Karhunen, K. "Über lineare Methoden in der Wahrscheinlichkeitsrechnung". In: *Annals of Academic Science Fennicae, Series A1 Mathematics and Physics* (1947).

[10] Klein, B. *FEM: Grundlagen und Anwendungen der Finite-Element-Methode im Maschinen- und Fahrzeugbau*. Jan. 2015. DOI: 10.1007/978-3-658-06054-1.

[11] Liao, Z., Hossain, M., Yao, X., Mehnert, M., and Steinmann, P. "On thermo-viscoelastic experimental characterisation and numerical modelling of VHB polymer". In: *International Journal of Non-Linear Mechanics* (Sept. 2019). DOI: 10.1016/j.ijnonlinmec.2019.103263.

[12] Loève, M. "Fonctions Aléatoires du Second Ordre". In: *Processus Stochastiques et Mouvement Brownien*. Paris: Gauthier-Villars, 1948.

[13] MacNeal, R. H. "A hybrid method of component mode synthesis". In: *Computers and Structures* 1.4 (1971), pp. 581–601.

[14] Muravyov, A. and Rizzi, S. "Determination of nonlinear stiffness with application to random vibration of geometrically nonlinear structures". In: *Computers and Structures* 81 (July 2003), pp. 1513–1523.

[15] Rayleigh, J. and Strutt, W. *The Theory of Sound, Vol. 1. 2nd*. The Theory of Sound. London and New York (first edition in 1877): Macmillan, 1894.

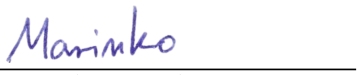
[16] Rubin, S. "Improved Component-Mode Representation for Structural Dynamic Analysis". In: *AIAA Journal* 13.8 (1975), pp. 995–1006.

[17] Rutzmoser, J. B. “Model Order Reduction for Nonlinear Structural Dynamics”. English. PhD thesis. Chair of Applied Mechanics, Technische Univeristät München, 2017.

Disclaimer

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

Garching, May 30, 2022



(Signature)